Assessing challenges of continuous integration in the context of software requirements breakdown: a case study

Master of Science thesis in Software Engineering

Adam Debbiche
Mikael Dienér

Chalmers University of Technology
University of Gothenburg
Department of Computer Science & Engineering
Gothenburg, Sweden 2014

Assessing challenges of continuous integration in the context of software requirements breakdown: a case study

Adam DEBBICHE

Mikael DIENÉR

Supervisor: Dr Richard Berntsson Svensson

Examiner: Dr Miroslaw Staron

Chalmers University of Technology
University of Gothenburg
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Acknowledgements

The authors of this thesis would like to thank the supervisor Dr Richard Berntsson Svensson for his invaluable feedback and the examiner Dr Miroslaw Staron. We would also like to thank Ericsson for giving us the opportunity to conduct this research on their premises. We thank our industry supervisor Claes Hansen for helping us throughout our work. Special thanks goes to Emil Öberg and Christer Brander Nilsson for making this thesis project possible!

The Authors, Göteborg 15/06/14

Abstract

The complexity of software development has increased over the last few years. Customers today demand higher quality and more stable software with shorter delivery time. Software companies strive to improve their processes in order to meet theses challenges. Agile practices have been widely praised for the focus they put on customer collaboration and shorter feedback loops. Companies that have well established agile practices have been trying to improve their processes further by adopting continuous integration - the concept where teams integrate their code several times a day. However, adopting continuous integration is not a trivial task since it requires frequent integration of smaller units of requirements that can be tested separately. This thesis presents a case study in which we, based on interviews at a major Swedish telecommunication services and equipment provider, assess the challenges of continuous integration and software requirements breakdown and their influence on the implementation of a continuous integration process.

*K*eywords- continuous integration; requirements breakdown; software; challenges

Contents

1 Introduction

Software organizations today face a market with ever changing requirements and pressure to release more often. The use of agile practices has increased because of the emphasis they put on customer collaboration and embracing change [32]. Still, companies have been looking into shortening the feedback loop further and releasing more often to the customer by embracing continuous integration [32].

Continuous integration relates to the frequency at which changes are checked in [50]. Each check in results in a new working build of the software provided that all tests are passed. Continuous integration emphasizes multiple code check-ins on a daily basis as opposed to nightly builds. It is the next step in the evolution of an agile R&D company that has established agile practices in place [32]. It has been suggested that continuous integration increases the frequency of software releases and shortens the feedback cycle [16]. Additionally, Miller and Carter [30, 31] state that continuous integration reduces the time developers spend on checking in new code while maintaining the same level of product quality.

When it comes to integration frequency, most teams have a fixed window during a sprint where integration takes place. For instance Launchpad [25] has a four week iteration where integration takes place between week one and three. However, adopting continuous integration entails integrating more often, possibly several times per day [14].

Lacoste [25, p. 387] reports on the difficulties of introducing a continuous integration system at Launchpad: "So by the end of week 3, everybody is rushing to squeeze in their remaining features and bug fixes. The integration queue is congested because running the whole test suite takes nearly 2 hours on a fast computer". Seemingly, this quote reports on difficulties related to integration queue congestion. However, one could question the reason behind last minute integrations. A team using an efficient continuous integration process should be able to treat the integration as a nonevent [14].

In order to support more frequent integration, requirements need to be small enough in order for developers to test them separately then integrate multiple times a day. This is not always a trivial task in the context of continuous integration [16].

Breaking down large user stories into small enough stories with the right level of detail and visible business and customer value has been identified as a challenge for the clothing retailer GAP after the adoption of continuous integration [16]. Similarly, the open source project FreeBSD also struggled with the break down of work tasks into smaller pieces when implementing Symmetric Multi-Processing (SMP) into the kernel using a continuous integration process [19]. This thesis, through a case study at Ericsson, explores the challenges of continuous integration and requirements break down and how the latter influences the implementation of a continuous integration process. Semi-structured interviews were conducted with a total of 13 subjects in both Sweden and China.

The remainder of this thesis is structured as follows: in the next section, we present the background and related work. In section 3, we describe our research approach. In section 4, the company where this case study was conducted is introduced along with how continuous integration and requirements breakdown are currently done. Section 5 presents the results of our data collection followed by the discussion in section 6. The thesis ends with a conclusion and a future work section.

2

Background and Related Work

The purpose of this section is to provide the background of this study and a review of related literature. First, the background is introduced followed by a related work section about agile methodologies, continuous integration, devops, continuous deployment and requirements breakdown.

2.1 Background

Continuous Integration has its roots in the eXtreme Programming (XP) agile method [25]. The goal for each developer is to commit new code several times a day then build and test the software. A successful test run results in a new build that the team can deploy. A continuous integration system often involves using a continuous integration server that automates the building process [14]. When a developer has added a feature or fixed a bug then the code is tested locally and built before being pushed to a continuous integration server. At this stage, the server merges the new code with the latest version on the server. A new version of the software is built provided that the merge is successful. Otherwise, the developer is directly notified of the conflicts.

Figure 2.1 shows the concepts of continuous integration and its relationship to other related concepts. These are further described in the next section. Lean software development's principles of eliminating waste has influenced agile methodologies which in turn has impacted DevOps as an emerging concept [21]. Similarly, software requirements breakdown supports agile methods in that customer features are broken down into user stories or scenarios to be implemented by developers. This becomes more apparent when adopting continuous integration [16, 19]. Continuous integration itself can be seen as an enabler of continuous deployment [32].

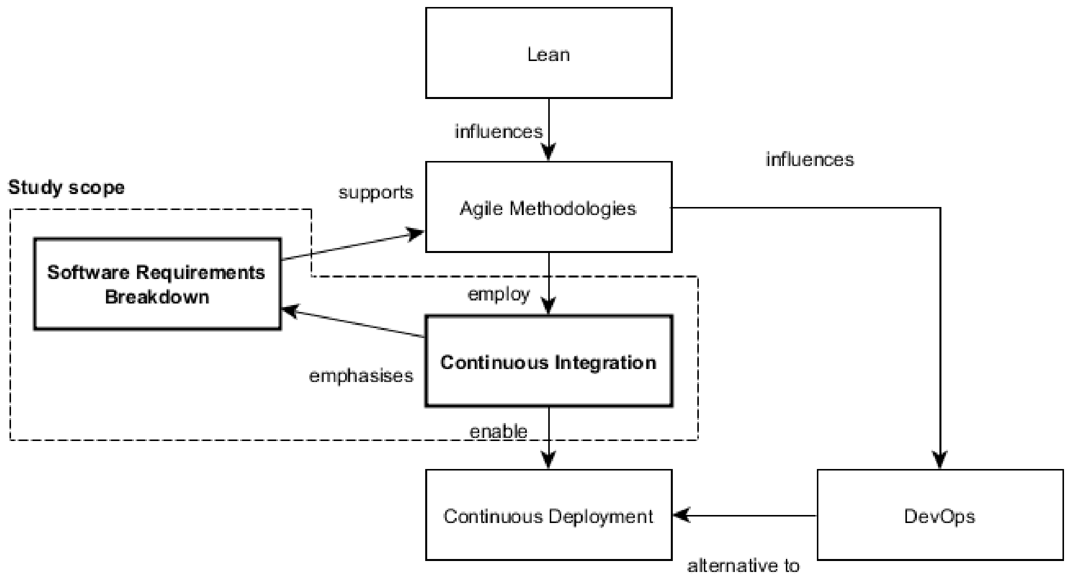


Figure 2.1: Concepts (e.g. processes, principles and methodologies) and their relationships.

2.2 Related Work

This section presents current literature related to the concepts in figure 2.1.

2.2.1 Lean Software Development

Lean is a software development process that has its roots in the Toyota Production System [36]. Toyota wanted to eliminate waste from its car production line and deliver the product as fast as possible to the customer. Lean is the adoption of Toyota's successful process to software development. It is guided by 7 fundamental principles [35, 36]:

- Eliminate waste: eliminate anything that doesn't add value to the customer and the product.
- Amplify learning: increase knowledge by constantly learning from customer feedback.
- Decide as late as possible: delay decision making in order to stay flexible and embrace change.
- Deliver as fast as possible: release to the customer often and without delay.
- Empower the team: engaging the team is important in order to unlock its full potential.

- Build integrity in: focus on quality assurance early on and embed it in the process.
- See the whole: identify the whole picture and not only focus on parts of it.

Lean software development principles are seen as compatible with the mindset of agile methodologies such as Scrum, Kanban and XP [35]. Additionally, recent software development practices such as continuous deployment have been used within lean software development [7].

2.2.2 Agile Methodologies

Agile development methods have been researched in previous literature due to their widespread use and increased interest shown by the software industry [34]. A systematic review of previous agile literature [10] revealed that agile methods have lead to increased productivity in teams, closer customer collaboration and higher quality software releases thanks to the faster discovery of bugs. Tacit knowledge transfer also improved when using agile practices such as pair programming [10]. Different success factors of adopting agile practices have been identified such as the need for senior management to have a clear strategy as well as offering ample support for agile deployment efforts [34]. Furthermore, a proper software process improvement should be undertaken where the organisation studies the impact of change and its impact on the development teams [18, 47] that are about to transition to agile.

The effects of introducing agile practices were also studied in previous literature. First, introducing Scrum in a team not only resulted in an increase in customer satisfaction but also made the team more efficient and decreased overtime hours [26]. In addition, developers' satisfaction was higher when using eXtreme Programming (XP) as opposed to a traditional development method. Moreover, the same developers enjoyed a better and more relaxed working environment [27]. Project management was also easier when using agile methods since they offer a higher degree of flexibility compared to plan driven projects which leads to higher quality releases [6].

According to the stairway to heaven model [32], Agile research and development (R&D) is the preceding step of continuous integration. Where the product development organization should have successfully adopted an agile approach to R&D in order to continue upwards in the *stairway to heaven*.

2.2.3 Continuous Integration

Fowler [14] presents a set of benefits related to the implementation of continuous integration. First, the risk of integration errors decreases since integration is no longer a daunting and unpredictable task that is done at the end of each sprint. Rather, a small one that takes place several times a day, meaning that integration is no longer postponed. Second, bugs are discovered and fixed earlier due to the frequency of builds and tests. Bugs are also easier to catch since each small change to the code is checked in and tested separately before being integrated into the mainline [14]. A typical continuous integration process can be seen in figure 2.2.

Despite the advantages of adopting continuous integration. Many organizations have faced a lot of challenges when migrating to it. Challenges related to moving from agile practices to continuous integration have been identified [32] : A company that specialises in embedded hardware had problems with handling component dependencies during development and integration. Automatic testing in the context of continuous integration was also a barrier since it involved testing code running on embedded hardware. The authors also note that the different tools available for continuous integration make it hard to know which ones to choose from.

The practice and implementation of continuous integration varies in industry [46]. This is due to the fact that the concept is interpreted differently. Sthål and Bosch [46] developed a descriptive model that facilitates the documentation of continuous integration practices and implementations. The model consists of an Integration Flow Anatomy which is essentially a directed acyclic graph (DAG) that shows the activities of a continuous integration process. The graph consists of two types of nodes: input (e.g. source code) and activity (e.g. executing test cases) [46]. Each node type has a set of attributes that outline its scope and characteristics. The purpose of the attributes is to describe and address the different variation points of continuous integration implementation identified by Ståhl and Bosch [46]. The model proposes different attributes related to activity nodes such as scope (testing, acceptance, deploying) and build characteristics (execution frequency and duration). The input nodes attributes presented are pre-integration procedure and integration target [46]. While the model has not been fully validated due to the small sample size, it has nevertheless been tested on a real project and the authors were able to identify areas of future improvement for the project and team [46].

Holck and Jørgensen [19] have researched the use of a decentralized continuous integration process in the open source community (FreeBSD and Mozilla) where developers are often distributed. The paper focuses on quality assurance with regards to the work processes, developer skills and work output. Submitting changes of good quality is used as the basis for granting commit access to outside contributors [19]. The authors found that developers working on open source projects are often free to pick any tasks or bugs they want to work on. Deciding when to integrate changes is also delegated to the developer. This is an advantage when compared to the more plan-driven work assignment process of traditional projects [19]. Breaking down tasks into smaller pieces is also prevalent in open source. However, breaking down large tasks is not trivial. For instance, adding support for Symmetric Multi-Processing (SMP) to the FreeBSD kernel resulted in multiple build errors and severely disrupted the work of other developers [19]. When it comes to using continuous integration, both FreeBSD and Mozilla have guidelines that any contribution needs to adhere to before it can be integrated into the mainline without breaking it. Both projects have an automated build system that verifies new commits. Any problems are quickly fixed and kept on the main trunk instead of being deleted [19]. Holck and Jørgensen argue that open source projects' ability to adopt a decentralized and bureaucracy free continuous integration process is mainly due to two factors: the developers' freedom to integrate their contributions as they see fit and the *"don't break the build rule"* [19].

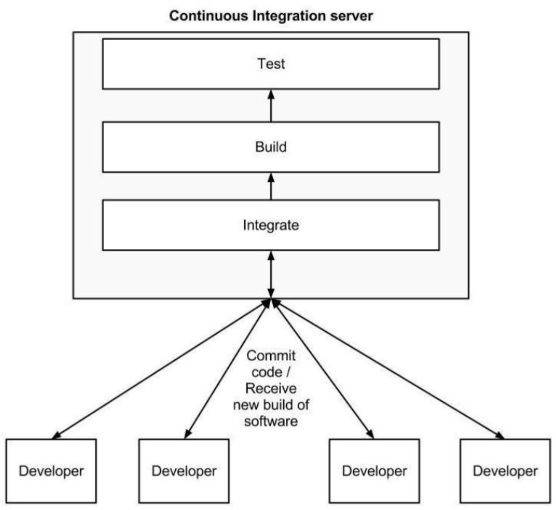

Continuous Integration server
Test
Build
Integrate
Commit code / Receive new build of software
Developer
Developer
Developer
Developer

Figure 2.2: The continuous integration concept [7]

2.2.4 Continuous Deployment

Continuous deployment complements continuous integration in an agile software development process. Indeed, Olsson et al. [32] argue that continuous deployment is the next step in their *stairway to heaven* model for a team or organization that has already successfully implemented continuous integration. Moreover, having a continuous integration server is seen as the pillar of a continuous deployment system [38].

Previous research argues that continuous deployment results in software being deployed faster to the customer [32]. Ries [38] notes that continuous deployment reduces cycle times with organizations deploying new version of their software up to fifty times a day [13, 51]. Furthermore, the company Atlassian points out that continuous deployment has increased the overall quality of their software products due to bugs being discovered earlier and being easier to fix since committed changes are often small [41].

Implementing a continuous deployment system is not a trivial task and does not happen overnight. Starting small with a simple continuous integration server, a version control system and a deployment script is enough according to Ries [38]. The deployment system will eventually become more elaborate over time as the entire process matures [13]. Indeed, IMVU has been working with continuous deployment for the past few years and is today able to run up to a million automated test cases a day [13].

2.2.5 Software Requirements Breakdown

Requirements are the basis of any software development project. The first step of the requirements engineering process is the identification of direct and indirect stakeholders. Failure to capture the correct stakeholders will result in incorrect requirements being elicited. This can negatively affect the project's outcome [43]. Once the stakeholders are identified, requirements are captured and documented. Different standards have also been presented for documenting requirements such as IEEE Std 830-1998. Elicited requirements can come from different sources, in different types (functional and non functional) and in varying degrees of complexity [17].

Requirements often need to be prioritized so that product managers know what to focus on and what requirements to include in a specific release. Previous literature proposes a number of models to assist in this task such as the cost value approach [22], the QUPER model [1] and Planguage [15].

Breaking them down to an appropriate level of abstraction is a challenging task. Indeed, effective requirements abstraction has been identified as a challenge in current literature that needs further research [37]. With regards to the different abstraction levels and its challenges, Gorschek and Wohlin [17] developed the Requirements Abstraction Model (RAM) with the aim of helping requirements engineers and project managers with handling requirements from different sources and abstraction levels.

The model has four steps: first, requirements are gathered from different sources. The requirements need to be understood by product management [17]. A number of attributes are used to help describe the requirements such as description, rationale, title and restrictions. The second step is placing the elicited requirements in one of the four abstraction levels: product level, feature level, function level or component level. The last step consists of working up or breaking down a requirement depending the original level of abstraction such as they obey two rules in the model [17]:

1: Any requirement needs to have a connection to a product goal

2: All the requirements need to be broken down to function level

The model does not consider necessary breaking down each requirement to the component level of abstraction [17]. When taken in the context of continuous integration, however, this could be an important step. In addition to that, the model stresses the importance of having adequate levels of abstractions which could be essential for breaking down requirements when using a continuous integration process.

Although earlier research has defined quality attributes for specifying unambiguous and testable requirements [3]. These quality attributes and standards do not guarantee that requirements have been broken down enough with regards to continuous integration. In fact, to the authors best knowledge, there is a lack of previous literature that studied large software requirements breakdown in the context of continuous integration. The problem has however been reported in earlier studies [16, 19].

As seen in previous literature [14, 32], both continuous integration and breaking down larger requirements has its own set of challenges. A team that uses an agile method like Scrum has a high degree of flexibility as to when integration ought to take place. For instance, a team with three week sprints can work easier with large requirements compared to a a team that has one week sprints. This is because longer sprints offers more time between implementation and integration which makes it easier to accommodate larger requirements. Such was the case for the Launchpad team [25] that integrated all of their work at the end of week 3 of each sprint.

Introducing continuous integration means each developer should integrate new code at least once a day but in practice it should be done *several times a day* [14]. This concept is a fundamental part of continuous integration. Regardless, having large requirements makes it very difficult to satisfy this condition unless a team has a well thought out process that allows them to break down large requirements without jeopardizing the core concept of continuous integration.

2.2.6 DevOps

DevOps is a term coined by Patrick Dubois back in 2009. It's a set of practices that advocate the collaboration between software developers and IT operations. It aims to shorten the feedback loop while aligning the goals of both the development and IT operations departments [21]. According to Hüttermann [21], DevOps emphasises the empowerment of people over processes, the need of automation in the development of new software, establishing quality measurements and creating a culture of sharing among people.

DevOps emerged to address the underlying disconnect between the development and IT operations departments [21]. Hüttermann [21] argues that conflict is inherently "baked" into this system where developers strive for change while IT operations strive for stability. Each department continuously improves its internal processes in order to become more efficient at achieving its goals.

The widespread adoption of agile methodologies has improved the performance of software developers teams [10]. DevOps widens the spectrum of agile methodologies by not only bringing together programmers, testers and quality assurance engineers but also the IT operations staff by fostering communication, collaboration and aligning objectives [21].

3

Research Approach

This section introduces the purpose and approach of this study. First, the research objectives and the the research questions are presented. Next, the methodology and the approach for collecting data are introduced. The section ends with the data analysis and validity threats sections.

3.1 Research Purpose

The purpose of this study is to identify the challenges of continuous integration and requirements break down and how the latter influences the implementation of a continuous integration process.

3.2 Research Questions

Currently, little research has been done on the challenges of adopting a successful continuous integration process. To the authors best knowledge even less has been done on software requirements breakdown within the context of a continuous integration process and the challenges that may arise. Current literature reports on problems that could emerge from the emphasis continuous integration puts on integrating new code several times per day. For instance, Lacoste [25] reports on difficulties while introducing continuous integration which in LaunchPad's case might be due to integration queue congestion. However, a team using a healthy continuous integration process should be able to treat integration as non-event and new code should be pushed every few hours without any obstacles [14]. Also, previous studies state that breaking down software requirements while working with continuous integration is a challenge [16, 19]. As a result, the following research questions were defined:

RQ1: What are the challenges of implementing a continuous integration process in practice?

RQ2: What are the challenges of software requirements breakdown in practice?

RQ3: How could the breakdown of software requirements influence the adoption of continuous integration?

3.3 Research Methodology

This thesis reports on a 6 month (January 2014 - June 2014) case study. The choice of methodology was based on the real-life context of this particular research case. Indeed, case study research is considered particularly suitable where the context under study could be difficult to control [29]. Case study research mainly results in qualitative data which is deeper and extensive in its nature, as opposed to quantitative research (e.g. survey and experiments) [40]. The purpose of case study research is to investigate a specific contemporary phenomenon [52]. This thesis was conducted as a single case study where the focus and case is an organisation (Ericsson, see section 4). The phenomenon being investigated is the adoption of continuous integration by teams working on the SGSN-MME product (see Section 4). Consequently, the unit of analysis is the teams under study that are adopting continuous integration. Moreover, this research was conducted with an interpretive perspective, as described by Klein and Meyers [23]. That is, to better understand social and organizational contexts where each individual's interpretation is of importance [23]. In addition, this study follows the explanatory-descriptive purpose as classified by Robson [39]. According to this classification [39], there are four different purposes of research, namely exploratory, descriptive, explanatory and improving. The explanatory-descriptive purpose does not only focus on describing a situation and how things work but also on finding causal relationships between problems and situations.

3.4 Data Collection

Semi-structured interviews [40] were used as the method of collecting data in this study. When performing case study research interviews are considered an important and common technique for collecting data [40]. The primary objective is to collect data based on the interviewees memory, opinion or impression [42]. Semi-structured interviews often include an interview protocol but are not necessarily followed strictly. Additionally, a balance of open and closed questions is used. That is, to allow for exploration and improvisation regarding the subject [40].

In order to identify relevant subjects for this study, the selection process was carried out with help from a ”gatekeeper” at the company under study. The presence of a ”gatekeeper” could improve the chance of ”gaining access” to the selected subjects [45]. Including different roles, personalities and teams is one of the goals when selecting subjects in order to get a better distribution of viewpoints. This is recommended due to the qualitative nature of case studies [40]. The selected subjects were invited by email through a common email template containing a short description of the purpose of the study along with a proposed date and time. A total of 13 interviews were performed for this study. More specifically, eight subjects from Sweden and five subjects from China. Table 3.1 contains more details about the subjects.

The interview protocol (Appendix A) was inspired by the work of Claps [7]. However, many questions were changed or added to fit the scope of this study. As suggested by Yin [52], a pilot interview with an employee relevant to the study, was conducted in order to check the validity of the interview protocol. The goal was also to assure that an interview session is not too long and more importantly that the questions would be both relevant and at an appropriate level of detail. Feedback from the pilot interview was used to make minor refinements to the interview protocol.

Interviews were conducted during working hours and were either performed at Ericsson (Lindholmen, Sweden) or via video conference. The interviews lasted between 25 and 65 minutes. Both researchers took part in all interviews. One asked the questions while the other focused on taking notes. The author taking notes also asked follow up questions in some cases. All interviews, upon permission, were recorded. That is, to secure a backup of interview details. Even though written notes were taken during the interviews, it's hard to capture every important detail [40].

During an interview session, the subject is first introduced to the purpose of the study and its objectives then guaranteed anonymity and asked for consent about recording. The interview is divided into three parts and follows the Pyramid model [40] which begins with specific questions and continues with open ended ones. The first part is about background and understanding. The goal is to make the interviewee comfortable and to establish trust. The second part contains more open-ended questions related to continuous integration. The third part is about requirements break down. The interview ends with two wrap up questions where the interviewee can add anything they feel is relevant to the research.

After each interview, extensive notes were taken while listening to the recording and consulting the written notes taken during the actual interview session. Taking extensive notes is an alternative to full transcription, where the goal is to pick out relevant passages, quotes and examples [39]. The purpose is to interpret the answer of the interviewee to each question. Once done, the extensive notes were sent to the subject for verification via email in order to make sure that the interpretation was accurate [40]. All subjects verified the written extensive notes with none or, minor clarifications.

3.5 Data Analysis

Case study research often results in large amounts of qualitative data. Especially when using interviews as the primary method for collecting data. In order to properly analyse qualitative data, specific techniques are commonly preferred [42]. The purpose of analysing collected data is to be able to draw conclusions in the end. Yin [52] argues that this needs to be done with a clear chain of evidence. That is, the reader must be able to follow all the phases from how the data was collected and analysed to how the conclusion were derived [40, 52].

While different approaches to qualitative analysis exist, there are certain recurring similarities such as coding and finding patterns [39]. Data analysis in this study was done using thematic analysis [4]. It follows a six steps method presented by Braun and Clarke [4]. Thematic analysis was chosen due to its flexibility and ability to extract important features from large sets of data [4]. It is often seen as similar, to some extent, to content analysis when extracting patterns. However, according to Braun and Clarke [4], content analysis usually focuses on smaller details while providing qualitative frequency count. The main goal is to identify themes and patterns from the collected data. In this case study, both authors took part in all the steps as described below:

Step 1 - Familiarize yourself with the data: The goal of this step is for the researchers to get acquainted with the data collected. For this study it meant translating interview recordings into extensive notes while interpreting the interviewee's answer to each question.

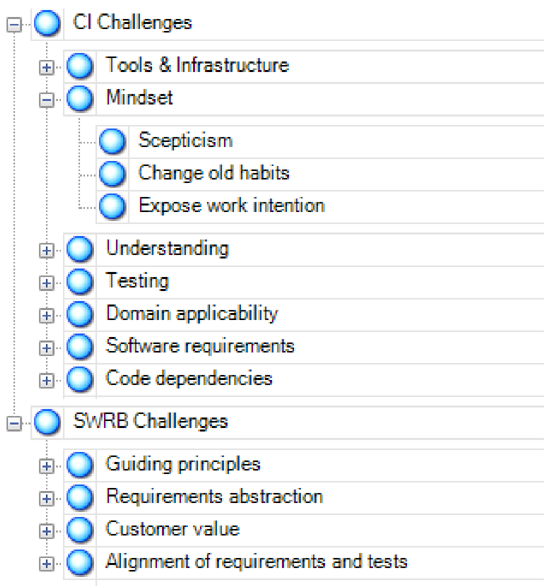
Step 2 - Generating initial codes: This phase involves creating the initial codes from the data set [4]. Qualitative data analysis software offers a systematic way of organizing, storing and visualizing collected data [39]. For this reason, NVivo (`http://www.qsrinternational.com/`), a specialised software for qualitative data analysis was used for the coding process in this thesis. Open coding as described by Robson [39] was used throughout the analysis phase since it is useful for categorizing data. For this study, the answers in the extensive notes obtained from the interviews were coded into categories, each piece of data coded can belong to more than one category. It is important to note that none of the categories were obtained prior to coding as this is highly discouraged when using open coding [39]. The extensive notes were imported into NVivo and then initial codes were generated. In order to get used to coding the data in NVivo, answers of simple questions from the interviews were initially coded from each of the 13 subjects. After that, the focus was on generating initial codes from questions 15, 24 and 32 since they represent a natural entry point to RQ1, RQ2 and RQ3 respectively. Finally, initial codes were extracted from the remainder of all thirteen extensive notes. See figure 3.1 for an actual view of the nodes in NVivo.

Step 3 - Searching for themes: This step entails classifying the codes initially generated into broader themes. This usually results in codes being grouped into themes and sub-themes [4]. For this study, the initial codes were put into three main categories based on the three research questions. Each main category was in turn divided into sub-categories containing sub-themes. Each sub-theme contained any related codes. Also, any codes that did not fit into any of the categories were grouped into a separate category in order to minimize the risk of disregarding potential new data leads.

Step 4 - Reviewing themes: In this phase, the themes identified in the preceding step are refined. This means that some codes might be merged if they are deemed too similar. Others might be moved from one category to another. For this thesis, the focus was on determining whether the identified categories were appropriate and reflective of the actual data. This was decided based on recurring patterns found in the data and whether they are supported by multiple subjects. That said, through an iterative process, new sub-categories were identified (alternatively renamed or removed). New codes were created as well as renamed or removed. The purpose is to get an accurate representation of the data set [4]. Figure 5.1. 5.2 and 5.3 show the output of this step in the form of thematic maps for RQ1, RQ2 and RQ3 respectively which is useful to demonstrate the relationship between codes [4].

Step 5 - Defining and naming themes: This step requires the scope of each main category to be clearly defined. This includes making sure that the codes included within are consistent. Braun and Clarke [4] argue that the 'story' of each theme needs to be identified. This step is essentially a refinement built upon the previous step. For this study, all the codes in each category were verified in order to make sure that they were consistent with the overall theme of the category. As part of this step, the final name of each theme was also decided upon before producing the results.

Step 6 - Producing the report: As the name suggest, the final step is to produce the final report detailing the results of the analysed data. The findings of the data analysis need to be produced in a concise, logical manner that is easy to follow in order to convince the reader of the points the study is trying to demonstrate [4]. Results of this study are reported in section 5.



- CI Challenges
  - Tools & Infrastructure
  - Mindset
    - Scepticism
    - Change old habits
    - Expose work intention
  - Understanding
  - Testing
  - Domain applicability
  - Software requirements
  - Code dependencies
- SWRB Challenges
  - Guiding principles
  - Requirements abstraction
  - Customer value
  - Alignment of requirements and tests

Figure 3.1: Screen cap of themes and sub-themes coded in NVivo as nodes

Team	**Number of months using CI**	**Average time between integrations before CI (in weeks)**	**Average time between integrations using CI (in weeks)**	**Role**						**Years of experience**					
										Organisation			Software Development		
				Software Developer	Line Manager	CI Driver	Agile Coach	Configuration Manager	Product Owner	1-3	3-10	10+	1-3	3-10	10+
T1	18-24	4-7	1	2	0	0	0	0	0	0	0	2	0	0	2
T2	2	12-24	1-2	1	0	0	0	0	0	0	1	0	0	0	1
T3	12	4-24	<1	1	0	0	0	0	0	0	0	1	0	0	1
T4	0	2-6	0	2	0	0	0	0	0	0	1	1	0	1	1
T5	2-3	4	1	1	0	0	0	0	0	1	0	0	1	0	0
T6	12	4-8	1	1	0	0	0	0	0	0	1	0	0	0	1
T7	8	4	<1	0	0	1	0	0	0	0	1	0	0	1	0
No team	N/A	N/A	N/A	0	1	1	1	1	1	0	2	2	0	1	3
Total	N/A	N/A	N/A	8	1	2	1	1	1	1	6	6	1	3	9

Table 3.1: Study subjects and team demographics - Years of experience is included as intervals due to confidentiality reasons.

3.6 Validity Threats

Qualitative research often receives much criticism by positivists, mostly regarding trustworthiness [44]. This is due to the opinion that qualitative research cannot address issues of reliability and validity in the same way as quantitative research. Dealing with these issues directly throughout the research is essential in order to ensure a measure of academical soundness [44]. This study has adopted the four aspects of validity (construct validity, internal validity, external validity and reliability) by Runeson and Höst [40] in order to address the issues of reliability and validity. These aspects were defined especially for qualitative research studies (e.g. case study research) and based on commonly used aspects of validity and threats to validity in literature. However, they choose to operationalise these aspects directly towards qualitative research.

3.6.1 Construct validity

Construct validity reflects to what extent the research methodology captures studied concepts and what is investigated according to the research questions [40].

The interview protocol was designed based on work done by Claps [7] and tested during one pilot interview. Minor revisions were made based on the feedback from the pilot interview. To reduce bias during the selection of subjects, a "gatekeeper" [45] at the studied company was used. That is the researchers did not influence the selection of subjects. In order to obtain a true image of the subjects' opinion, absolute anonymity was guaranteed, both within the company and externally. Key concepts, such as continuous integration, were clarified to minimize the risk of different interpretations between the researchers and the subjects. To minimize the risk of subjects having different interpretations or guessing the purpose of the study, a common introduction manuscript during the interview was used. This manuscript contains the purpose of the study, an anonymity clause and some general information about the interview and its structure.

Triangulation is important within empirical research, especially when performing research that primarily relies on qualitative data [40]. For this reason, triangulation has been kept in mind when performing this research, for instance, interviews have been performed with subjects from different teams, roles and countries to triangulate data. However, a limitation to this study is the use of a single method for collecting data. On the contrary, the case under study did not include much documentation that would benefit this particular study, which left interviews as the main source of data.

Another limitation of this study is that it only included thirteen interviews, all within the same company. However, they were performed until a saturation point was reached. This means that relatively little new information would be discovered if further interviews are to be performed.

3.6.2 Internal validity

Internal validity refers to the risk of interference in causal relations within the research. For instance, if the researcher investigates whether factor A affects factor B there is a risk that factor B is also affected by factor C. Researchers need to pay close attention to alternative factors that might affect the investigated factor, otherwise there is a threat to internal validity [40].

One of the objectives of this research is to find a causal relationship between challenges of breaking down software requirements and challenges of adopting a continuous integration process. Since the challenges of adopting a healthy continuous integration process can be caused by numerous other factors than the challenges of breaking down software requirements, this research also investigates the existence of other factors that might be an impediment to adopting continuous integration.

The process of breaking down software requirements was undergoing a major overhaul during the time of the study. This meant that there was a risk of misunderstanding the context for which a statement had been provided. To mitigate this risk, follow up questions were asked to clarify the context of the said statement.

3.6.3 External validity

External validity refers to what extent the findings of the research can be generalised and of interest to other cases, such as other companies or individuals outside the study [40].

This study included interviews with different roles and teams to benefit the external validity of this research, on the other hand, only performing interviews at one company affects the generalisability of the results negatively. However, the purpose of qualitative studies puts more focus on describing and understanding a contemporary phenomenon and less on the ability to generalize the findings beyond the boundaries of the studied settings [2]. Nevertheless, results from this study may benefit the investigation of phenomena within similar contexts [2].

Context and characteristics of the teams and subjects included in this research can be found in Table 3.1. This to simplify the process of identifying the context for which the findings are relevant. However, replicating a research of qualitative nature is often difficult, due to the fact that identical situations cannot be reproduced [2].

3.6.4 Reliability

Threats to reliability relate to what extent the findings of the study are dependent on the researchers that executed the research. For instance, to what extent can the study be reproduced with similar results [40].

This study has been performed by two researchers which increases the reliability and reduces the risk of single researcher bias. In addition, all research design artifacts (e.g. interview protocol), findings and each step of the research have been peer reviewed by an external researcher (university supervisor).

All data collected and documented, such as interview notes, audio recordings, in this study were digitalized and stored. This in order to allow for future external assessment. In addition, this report will try to provide a clear chain of evidence throughout the research to further increase the reliability of the study [40].

As suggested by Robson [39], no pre-determined codes were applied to the collected data. Thus avoiding forcing the data into a specific code or category. Codes were instead induced from the data collected.

4

Case Study

This section provides a more in depth view of the company studied in this thesis work.

Ericsson AB is a world leading provider of telecommunication equipment and services. It supplies solutions to telecoms operators, government agencies, media, transport and automotive industries. Ericsson offers a wide range of products such as base stations, radio access networks (such as GSM and LTE), microwave networks as well as products for television and video such as content and video management. Ericsson has more than 100.000 employees and offers its services to customers in 180 countries [12].

This case study was conducted within the Serving GPRS Support Node Mobility Management Entity (SGSN-MME) program. Meaning the study was done in collaboration with employees working on the development of the SGSN-MME product [11]. SGSN-MME is a product developed by Ericsson that consists of both hardware and software modules. It is essentially used to deliver packets of data through networks (such as GSM, WCDMA and LTE) [11].

4.1 Continuous Integration at Ericsson

Ericsson has heavily engaged in improving its approach to software development amid the increasing competition in the market. This is evident in the endeavors undertaken to transform the company from a plan driven organisation to a more lean and agile one [33]. A number of cross functional teams (XFT) working on the SGSN-MME [11] product have adopted continuous integration as part of their development process, albeit with a varying degree of maturity.

The teams working on the SGSN-MME product use multiple branches for development and integration with different quality assurance policies (see figure 4.1):

- **Work branch (WB):** used by team members locally first when a new feature is being developed.

- **Latest Local Version (LLV):** this branch includes tests that cover expected changes in functionality.
- **Pre-Test Build (PTB):** full regression tests are run on this branch by.
- **Latest Stable Version (LSV):** complete system tests are run on this branch.

Once a team begins working on a new feature or a bug fix, changes are first pushed to their work branch. When a feature is finished then the changes are pushed to the team's LLV where the new functionality is tested. If the LLV tests are successful then the new changes are delivered to the PTB branch where full regression tests are run. Finally, the new code is integrated with the LSV branch where the system is tested as a whole.

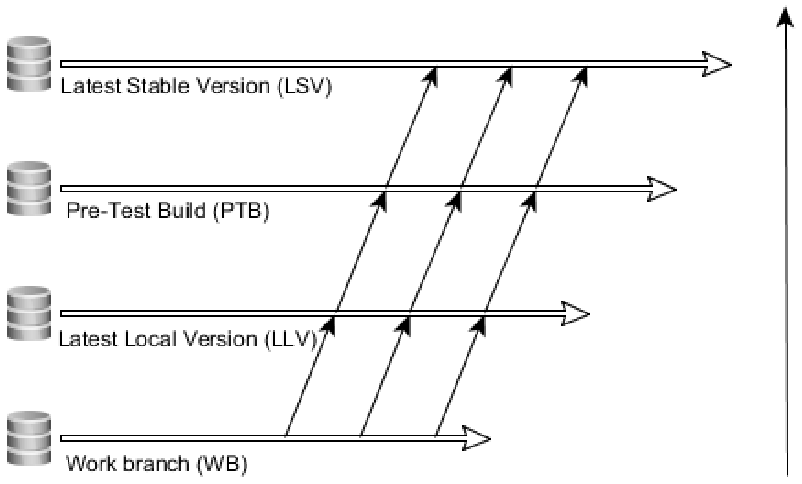


Figure 4.1: Illustration of the continuous integration branch strategy used at Ericsson. Note the stricter quality assurance measures between different branches.

The quality assurance policies of the branches listed above increase as illustrated by figure 4.1. Each branch has more rigorous tests that new code needs to pass in order for it to be accepted. For instance, integrating code to the PTB branch requires that the code to first pass the tests of the LLV branch. Likewise, the LSV branch has even higher quality assurance tests in place than PTB. Any new features or code integrated to the LSV branch must be of high quality that can be delivered to the customer. Therefore only stable changes are committed, unexpected and unknown bugs should not show up (only known issues).

Note: The Pre-Test Branch (PTB) was subsequently removed, however the concept and function of the branch remains and is fulfilled by other branches.

4.2 Requirements Breakdown at Ericsson

Ericsson has thousands of requirements that come from different sources. Some requirements stem from international telecommunication standards and specifications such as 3GPP while others are from customers. IBM Rational RequisitePro is sometimes used to document and keep track of requirements.

In recent years, Ericsson has introduced the concept of cross functional teams (XFT) and operational product owners (OPOs) as part of its agile transformation program. The OPO's task is to work with the feature product backlog and decide what requirements should be implemented and in which order (when). A cross functional team uses this as input to create the sprint backlog (typically during sprint 0). It is largely up to the cross functional teams to decide how their requirements (for a sprint) are broken down. System managers have previously been responsible for breaking down requirements and conducting pre-studies. This has however changed and now system managers are more integrated into the teams.

5

Results

This section presents the findings of this study after the data was analysed according to the method by Braun and Clarke [4] (see Data Analysis).

The structure is divided with the aim of clearly answering the research questions presented earlier. First, the challenges of adopting continuous integration are presented as answer to research question one. Next, the challenges of breaking down software requirements are laid out (research question two). The last section deals with research question three and how the challenges of requirements breakdown relate to the adoption of continuous integration. The challenges in section 5.1, 5.2 and 5.3 are structured according to their respective thematic maps (see figure 5.1, figure 5.2 and figure 5.3). Each theme (e.g. Mindset) illustrated in the 3 thematic maps has a respective source count. It reflects the number of interviewees that have mentioned said theme during the interviews. The source count is calculated by aggregating the individual source counts of each sub-theme (e.g. Scepticism).

5.1 RQ1 - Continuous integration adoption challenges

This section answers the first research question related to the challenges of adopting continuous integration in an organisation. The identified challenges are shown in the form of a thematic map in figure 5.1.

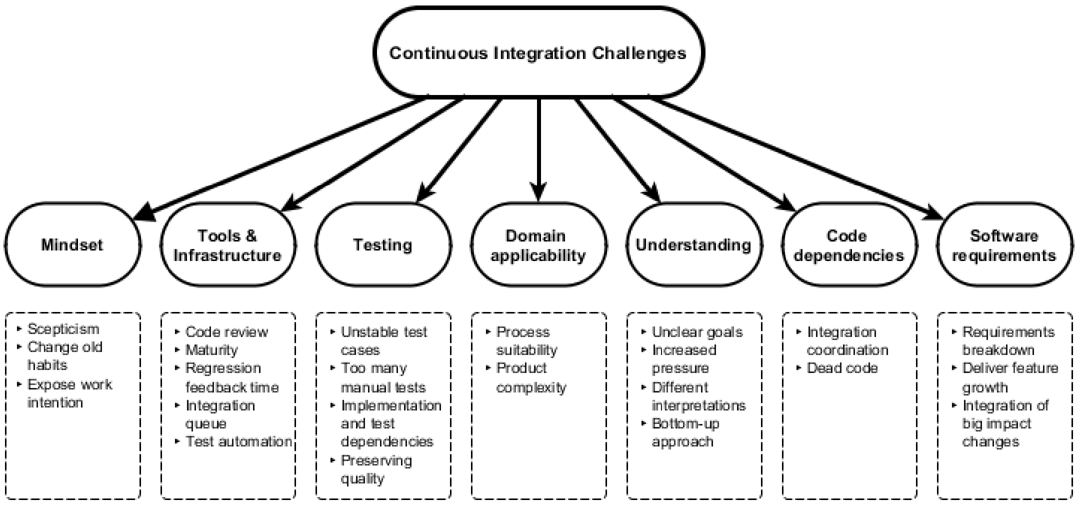


Figure 5.1: Thematic map of continuous integration challenges

The aggregated source count for each theme is as follows: Mindset: 10, Tools & Infrastructure: 13, Testing: 9, Domain applicability: 11, Understanding: 10, Code dependencies: 5 and Software requirements: 8. More details regarding source and reference count for each theme and sub-theme can be found in Appendix B.

5.1.1 Mindset

The results of this research found that challenges related to the developer mindset play an important role when transitioning to continuous integration.

Scepticism One of the mindset challenge identified at the studied company is being convinced about the benefits that come from adopting continuous integration. Interviewees (7 out of 13) report that they are positive about the concept of continuous integration but need to, by themselves, experience the benefits that comes with the change. This to be fully convinced and furthermore promote the adoption. Introducing such a big process change to about 30 teams can be a challenge, as described by a CI driver:

"*In the summer of last year, when we started to pilot CI we introduced this change to the team. However, not all of the people are buying the change, not all of them like this change. So we just had a very small group, like 4 designers. They started to pilot first, these four designers, I think they are the designers who liked challenge or liked the change so we started the pilot first to know what we should do, define the process and everything. And then, we later adapted this to the whole team. Maybe, the first introduction to the team is not very smooth because not everyone wanted but after the small group has already defined the draft version of the practice* *and later on it's much easier. People just follow.*" - CI Driver

Some developers that were interviewed are not fully persuaded about the benefits that continuous integration might bring to the SGSN-MME program described by agile coach:

"*From the very beginning, the team they question about the value for that, how much benefit we can get from the CI because actually, SGSN-MME we have already the streamline and we have already monthly delivery and they just want to know what's the benefit, if we want to improve delivery frequency to a weekly delivery. what's the benefit for that. That's the vision, the value we should introduce to the team.*" - Agile Coach

Similarly, a CI Driver involved in coaching and helping teams with transitioning to continuous integration highlights the importance of questioning a change:

"*I don't see the reason why we should use CI because of CI, it should always be beneficial to use it, not be a rule.*" - CI Driver

Even though the interviewees report on issues related to mindset and experiencing the benefits of continuous integration, some think this challenge could be overcome, as put by one software developer:

"*Sprint delivery is once a month, but CI is deliver as quickly as possible and I think only need two persons mindset change then we can deliver it as quickly as possible, don't think it's very hard for Ericsson to adopt CI.*" - Software Developer

The fact that many developers are questioning the benefits of continuous integration has led to more efforts being done in order to convince people, one software developer reports on work being done to increase the experience regarding the benefits:

"*I think all the teams, both from Lindholmen and Shanghai, we have discussions about how to do CI and how to let teams think it's very good and it can bring benefits to us.*" - Software Developer

Interestingly, a developer from one of the pilot teams, report that their team struggled with mindset related problems, mostly regarding seeing the benefits, and that they have now overcome these issues. This was done by a step-wise approach and constant team discussions regarding benefits.

Change old habits Developers often get used to working in a certain way. The introduction of continuous integration challenged those habit. Consequently, some developers found it hard to give up their old habits. A line manager mentioned that people that have been working at the company for a long time are harder to change compared to junior people because they got used to working in a certain way.

A developer part of the continuous integration pilot group stated that adopting continuous integration is a big change for SGSN-MME. Therefore, changing people's mindset and habit is a challenge that takes time. Before, developers could work on their code and deliver it at the end of the month. Presently, they are forced to think about integrating, checking regression results, reviewing code and writing test cases more often.

Exposing work intention Continuous integration emphasises early and frequent integrations. As a result, developers are compelled to expose their work earlier. Some interviewees (4 out of 13) found this to be challenge because they were used to big bang integrations at the end of a sprint. This gave them enough time to polish their code before integrating it. With the adoption of continuous integration and the increase in integration frequencies, developers are worried about integrating low quality code that could be questioned by experts and managers, as an agile coach put it:

"*Some teams they are not familiar or used to frequent delivery, because they feel safe if they can deliver once a month because they can make everything ready, if they have some changes, he can correct it on his own branch, don't have to deliver to the main branch and then everybody can see your faults right. It's a mindset change, and actually for the CI, we want to be open minded with your code, even to your code, and you should be confident to show your code to others even you have some faults...it's mindset change.*" - Agile Coach

Developer confidence plays a role in this issue. Teams that are more experienced working with continuous integration seem to be more comfortable about exposing their work earlier as stated by a configuration manager:

"*Some of the teams have gotten into it and are used to it, but of course for new teams it's kind of a different mindset to actually deliver code that you know is not ready..., that's the mindset* " - Configuration Manager

The line manager interviewed for this study argues that the shift in habits requires that developers be given enough time to adjust to the new ways of working with continuous integration.

5.1.2 Tools & Infrastructure

The tools and infrastructure supporting the continuous integration process are developed and maintained in-house at the studied company. These include tools for reviewing code, visualising regression test results, running automated test suites, checking in code and such. Below, the challenges of said tools are presented.

Code review: Tools for reviewing integrated code has been reported to lack the necessary features for supporting an efficient continuous integration process. For instance, visualisation of the "bigger picture" while performing a code review has been requested. However, current code review tools only support visualisation of each integrated change separately. This limits the ability to see the impact of smaller changes and how they related to the "bigger picture".

In Shanghai, Technical Area Responsibles (TAR) are responsible for performing code reviews. However, due to their limited number and availability this has become a bottleneck in the continuous integration process. In addition, the current limitations of code review tools mentioned earlier further complicate this matter.

The increased integration frequency, as a result of the ongoing adoption of continuous integration, has increased the number of code reviews needed. This in turn has increased the daily workload and pressure, especially for the TARs in Shanghai.

Maturity: The maturity of the tools and their surrounding infrastructure that developers use when integrating code has been reported as a challenge. These tools are often seen as not ready for an efficient continuous integration process. The infrastructure team responsible for developing and maintaining the tools currently has a long backlog that needs to be addressed in order to improve support for a smooth integration process.

Currently, it takes a long time for developers to integrate their work, especially to the PTB and LSV branches. This in turn prevents them from reaching the desired integration frequency. For example, it takes a considerable amount of time to rebase (updating local repository to the latest version) from the main branch when integrating new code. Similarly, the build framework and the delivery tools are lacking in terms of maturity and new ways for developers to integrate their codes need to be developed, as one configuration manager puts it:

"*After years of developing a product, process and tools tend to be tightly intertwined. When new processes come along then you need to adopt the tools for it.*" - Configuration Manager

This highlights the challenge that the introduction of continuous integration has placed on the tools used at the moment. They need to be better adopted to facilitate continuous integration.

Regression feedback time One common opinion at the studied company is that the feedback loops from the automated regression tests are too long. Regression feedback times are reported to take anywhere from four hours to two days, depending on which part of the product you are working on. The long feedback loops take valuable time and adds pressure to everyday work. One software developer expressed frustration regarding the long feedback loops:

"*...all my changes were done on Wednesday and I started a build Wednesday afternoon and at night, this Friday almost two days later I got a mail... that all the regressions were all passed, that's not CI.*" - Software Developer

This highlights the problem of getting feedback from regression tests up to two days after integrating code. By then, it could already be out of date. One software developer, stated that fast feedback loops are important when adopting continuous integration:

"*Fast feedback loops are, very important for CI to work we discovered.*" - Software Developer

This statement stresses the importance of fast feedback loops within a continuous integration process. This in order to fully derive the efficiency benefits that could come with the change of process. Nevertheless, long feedback loops, in contrast to "big bang integrations" are still preferable as mentioned by one software developer:

"*I think that doing it every day actually is better than doing it in a big bang as we used to do.*"

Integration queue The process of managing the integration queue has been difficult due to the nature of the SGSN-MME product. There are hundreds of developers working on common and different parts at the same time from different locations. This means that new code is constantly added to the integration queue. Consequently, two issues emerge. First, keeping track of all the deliveries while preserving quality becomes difficult. Second, the chance of blocking the integration queue increases due to the surge in integrations which can lead to the branch being blocked for several days. These two challenges manifest themselves primarily when integrating to the Pre-Test Branch (PTB).

Test automation The support of test automation is lacking in the current infrastructure. This makes maintaining the quality of the product difficult according to one developer. There are a lot of manual steps involved in integrating new code. Some system tests are run manually. The ability to automate tests is highly sought after by developers when checking in new code.

How to make system tests automated is still an open issue according to a CI driver interviewed, although a team has started looking into developing a new testing framework that better supports automation.

5.1.3 Testing

Challenges associated with testing at the studied company are the lack of automated tests along with a stable test framework according to developers.

Unstable test cases Test cases at the studied company are sometimes unstable and may fail regardless of the code. This makes the evaluation of the results difficult. The varying test coverage between different branch levels, for instance the Last Local Version (LLV) and the Pre-Test Branch (PTB), further complicates the evaluation of the integrated code. Since the PTB covers more tests that are stricter, teams may not able to guarantee that integrated code will not break the PTB branch, based solely on LLV test results.

Too many manual tests Automated tests are considered a prerequisite for continuous integration according to an interviewee. The current amount of manual tests is an obstacle to the efforts of adopting continuous integration. Many developers state that there are a lot of tests that need to be run manually before the code can be integrated. This resulted in gaps in the current testing frameworks.

The problem is more prevalent on the platform level (e.g. link and routing) of the SGSN-MME product. This means that teams working closer to the hardware suffer from more manual tests compared to the teams working on the application level.

Implementation and test dependencies A problem related to writing test cases is syncing them with the code they are supposed to test. Often, code is implemented before its test cases are written (or vice versa). This makes it complex to coordinate the integration of new code with its corresponding test cases. Developers don't always know what to do with new code that has no corresponding tests yet or test cases without implemented code (if using test driven development).

Preserving quality Maintaining the right level of quality while adopting continuous integration has been a concern. According to a CI driver, there has been too much focus on how how to introduce continuous integration but not so much on how to retain the quality of the product to be delivered. Additionally, the large amount of people working on the SGSN-MME product means that guaranteeing the quality of the increased integrations becomes challenging.

Finding an appropriate integration frequency without threatening quality is currently an issue. This is closely related to another identified challenge while adopting continuous integration, more specifically the increased pressure regarding higher integration frequencies (see 5.1.5). Pushing the integration frequency goal too eagerly could jeopardise the quality of each individual integration.

5.1.4 Domain applicability

The challenges related to the suitability of continuous integration at the studied company and the related product are presented in this sub-section.

Process suitability While taking a step towards more frequent integrations using continuous integration, the studied company has been experiencing problems using the same desired frequency throughout all parts of the product. Additionally, some subjects (6 out of 13) are of the opinion that continuous integration is not feature, domain and task independent. Meaning, the possible integration frequency differs depending on what kind of work being carried out.

Continuous integration might not be invented with all types of domains or industries in mind, as one line manager puts it:

"*...CI is not invented for all types of industries, but the general concept is good...*"
- Line Manager

Several software developers highlight the problem of using continuous integration for all parts of the product and features, with one stating that:

"*It depends a bit, I guess for the application parts, we have a lot of application teams here also, I guess for them it's a bit easier, when they have one main flow that they deliver, and then all the exceptional flows they can deliver one after another. For the platform teams, yeah as I said, for some features they might break it down to smaller deliverable packages, but not always. Like when they introduced a new version of the Linux Kernel. I mean, you cannot do that in small parts.*" - Software Developer

Another identified challenge related to the suitability of continuous integration is the ability to break down software requirements (see 5.1.7), as one software developer described:

"*Other challenges are...when you work in the platform area, it's often hard to break down one feature in many deliverable work tasks, if you for example change the whole communication on the "backplane", that you are using..., that was one legacy feature, you cannot deliver small parts of it, either you use... on the "backplane" or you don't*" - Software Developer

Product complexity SGSN-MME is a mature product that has been developed for over 15 years with around 30 teams currently working on it. This complexity is something that the developers struggle with when transitioning to continuous integration. As a result, many think that continuous integration is difficult to adopt in regards to the product and that it cannot be followed by the book. Compared to smaller products, where all code is merged to a single branch, the SGSN-MME development makes use of many branches which adds to the complexity.

Some believe that the complexity is due to how the SGSN-MME is designed: some changes (sometimes minor) require the node to be rebooted which is not appreciated by the customer. The use of quick workarounds rather than fixing the main problem also contributes to the complexity of the product.

While the size of the organisation, product and people involved in it has been recognized as a challenge by most interviewees (7 out of 13). Some think that the complexity of the product and the difficulty of transitioning to continuous integration is more related to the confidence of the teams and the tools at their disposal.

5.1.5 Understanding

Introducing continuous integration is not a trivial task. Results indicate that teams and management interpret the concept differently. This resulted in some challenges being identified as described below.

Unclear goals Setting up clear goals for the teams migrating towards continuous integration is currently an impediment for the SGSN-MME program. Pilot teams are used to explore the possibilities of working with continuous integration. These pilot teams later help other teams migrating. How teams adopt continuous integration has been up to them. While this freedom is generally welcome, some interviewees (5 out of 13) still believe that the overall goals are unclear. One line manager thinks that there are often some differences in how teams work with continuous integration which can lead to coordination problems and that a more general way ought to be developed.

Some developers indicated that they want the organisation to provide clearer instructions on how to proceed with the adoption of continuous integration. Another CI coach believes that more feedback from the pilot teams is needed before any clear goals can be established while emphasising that the aim should be to maximize integration frequencies while enabling teams to set their own pace and goals.

Increased pressure The initiative to adopt continuous integration has resulted in increased pressure on the teams according to some interviewees (4 out of 13). Despite the positive support and attitude towards the concept of continuous integration, teams feel that management would like it to happen faster than currently possible which leads to increased pressure. Some developers feel that they lack the confidence and experience to reach desired integration frequencies.

This increased pressure could lead to lower quality. One developer believes that focus should be on quality of the integrations and not the frequency as stated:

"*My understanding is: I agree, totally agree with CI, just the issue of how often we do the delivery, we do the integration. I think in different products, in different organisations, in different ways of working, we might need different frequency, we have to accept that reality, sometimes we can't deliver, integrate that often. It's OK, we just try to continuously improve*" - Software Developer

There seems to be a general consensus among developers that transitioning to continuous integration carries risks and a period of chaos and increased pressure. Hence, the frequency of integrations and how to proceed should be done in steps in order to minimize the risk of increased pressure.

Different interpretations According to a line manager, most managers have their own understanding of what continuous integration is and should be. There is also a difference in what the goal with adopting it is. For this reason, managers risk misunderstanding the concerns of their teams. One developer also thought that this disparate understanding is not limited to the managers but is also present among teams.

Bottom-up approach The findings of this study indicate that continuous integration is being implemented with a bottom up approach as stated by a CI driver:

"*We haven't really done this as a top down method...we are growing from beneath instead...we have been on management meetings and convincing the management .*" - CI Driver

A pilot team was initially established to pilot the continuous integration concept. Currently, the pilot team members act as CI drivers. They provide assistance and training through meetings (both formal and informal), workshops and discussions to the other teams within the SGSN-MME program. It seems that both management and the teams are mostly happy with the work done by the pilot teams. Most think that they are committed to helping other teams transition to continuous integration. However, the bottom up approach seems to have led some to believe that management could be more involved in the overall process of transitioning to continuous integration.

5.1.6 Code dependencies

A consequence of more frequent integrations when adopting continuous integration is that work needs to be divided into smaller pieces. This could mean that work that would otherwise benefit from being developed in one single integration, now might need to be split up into several integrations. Additionally, by dividing work into several integrations, development might be carried out by several developers instead of a single one. This stresses the importance of considering code dependencies and how this affects the integration process as demonstrated by the challenges below:

Integration coordination The results from this research indicate that the task of coordinating integration dependencies has been more difficult since the adoption of continuous integration. Consequently, four different issues have been reported by developers:

- Component interfaces need to be more clearly defined.
- It is harder to locate the source of errors during integration, because code is delivered from different teams.
- More failures have been experienced during integration.
- Need to wait until other components/parts are done before integrating work.

It has been suggested that a solution to some of the issues presented above could be the use of ”dead code”, which is described further in the next section.

Dead code Integrating partial code for a feature, user story or delivery is currently an issue for the studied company. This due to the testability of such integrations. Tests will fail until all parts are in place. A suggestion presented by multiple developers could be to create a test framework that allow integration of so called ”dead code”. Meaning, code that is activated and tested when all parts are in place. However, making code support this type of activation/deactivation might be more costly, according to one software developer.

5.1.7 Software requirements

Software requirements have been themselves identified as a challenge when adopting continuous integration at the studied company (however, challenges of breaking down software requirements is a research question in its own, see 5.2). It has increased the frequency at which teams integrate their code. Consequently, requirements that previously could be integrated on sprint basis now need to be broken down to allow more frequent integrations.

Requirements breakdown Results from this research indicate that the increased integration frequency has put further pressure on the task of breaking down software requirements. For instance, interviewees (7 out of 13) report that since the adoption of continuous integration, breaking down software requirements to enable more frequent integrations, has been challenging. These challenges are related to finding a balance between size, testability and assuring quality on the integration line. In addition, one developer reports on the lack of experience, constant re-prioritisation and new decisions on requirements, which makes the task of finding a balance even more challenging.

One developer reports on system tests for the SGSN-MME product being too big or too old which is an impediment to the process of breaking down software requirements.

Another challenge related to software requirements and their breakdown is the increased need of multiple copies of team and product branches. Results indicate that before the introduction of continuous integration only one team branch (LLV) and product branch (PTB) was needed.

Deliver feature growth One of the issues reported by developers with breaking down requirements when using continuous integration is delivering feature growth. It is difficult to know whether small changes that do not directly add value to a feature are worth integrating.

Some developers feel this is inevitable. For instance, sometimes you need to refactor code or make minor changes. These changes do not necessarily contribute growth to the feature itself but are still needed. This means that teams need to be prepared for integrations that do not automatically add feature growth to the customer per say. One developer stated that there is a trade off between creating customer value and integrating.

Integration of big impact changes One developer has identified the integration of changes with significant system impact as a challenge due to the high integration frequencies. This problem mainly concerns integrating work from the team branch (LLV) to the product branch (LSV/PTB).

5.1.8 Summary

Results show numerous challenges hinder the adoption of continuous integration such as presented in this section. It seems that the mindset as well as the tools and infrastructure are the two most mentioned categories by the interviewees (see Appendix B). People's mindset and attitude towards continuous integration plays an important role on the likelihood of them embracing it. Similarly, the tools, infrastructure and testing in place has a major impact on the success of the practical implementation of continuous integration.

5.2 RQ2 - Software requirements breakdown challenges

This section answers what challenges organisations might encounter while breaking down software requirements. This question is related to the second research question of this study The results from the thematic analysis (see Section 3.5) performed in order to answer this research question is illustrated in Figure 5.2 and described further throughout this section. Statistics regarding responses can be found in Appendix B.

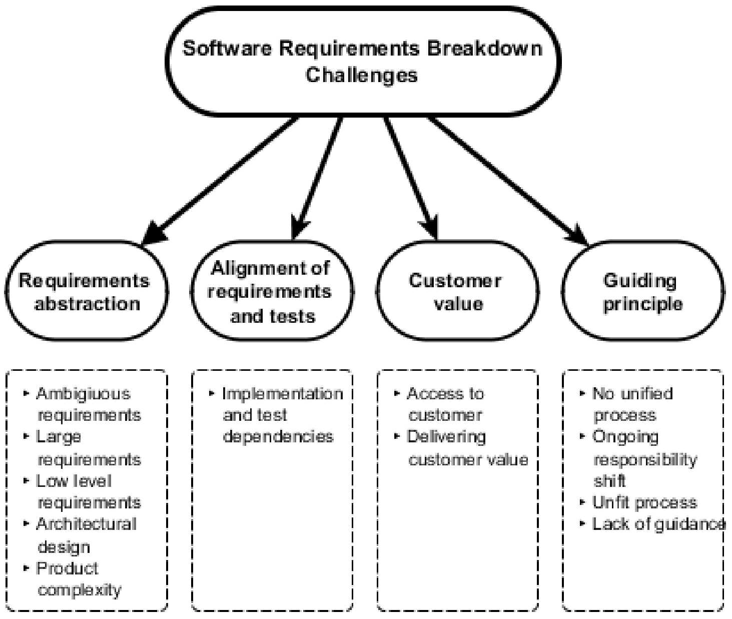


Figure 5.2: Thematic map of continuous integration challenges

The aggregated source count for each theme is as follows: Requirements abstraction: 12, Alignment of requirements and test: 4, Customer value: 7 and Guiding principle: 11. For further details regarding source and reference count for each theme and sub-theme see Appendix B.

5.2.1 Requirements abstraction

The results of this case study show that requirements abstraction was a challenge for developers. They struggled with requirements that are either too big, ambiguous or too low level.

Ambiguous requirements One challenge that the studied company has faced is ambiguous requirements. Requirements are reported to be either too big or too small and tend to change throughout time. In addition, the level of detail and quality of requirements vary depending on the type of requirement. For instance, requirements from standards (e.g. 3GPP) can be more detailed while requirements specified by the studied company often contain less detail. Consequently, requirements are very open to different interpretations depending on the reader. To get all details in place could be very time consuming and involve a lot of people, as described by a software developer:

"*...everything is pretty chaotic I would say, so before you always got clear requirements, you got EP (early phases presentation) where you knew what to do. Right now we get mails with like three sentences and everything is very fluffy and then you're working on prototype, you need a lot of feedback always from system management, system architects and so on, to find out what they really want, so that's our main problem right now in breaking down requirement.*" - Software Developer

The ongoing change mentioned by the software developer above is also identified as a challenge and described further in section 5.2.4. Some believe that the ambiguous requirements make the break down of software requirements more complex and time consuming.

Architectural design Some interviewees (4 out of 13) report that keeping the architectural design in mind, while performing and integrating work continuously, to be challenging. That is, since focus is on how to support more frequent integrations, the bigger picture could sometimes fall out of scope. As a consequence, people tend to take shortcuts and not follow a unified architectural design. This in turn, leads to difficulties in estimating the impact of changes. One software developer elaborates on the relationship between the adoption of continuous integration and not being able to see the impact of changes. This might lead to unexpected rollbacks from the product branch (LSV/PTB), which further can affect the time between integrations.

Another software developer states that breaking down software requirements tend to be easier at the end of a feature when the main architecture is in place. Similarly, a software developer argues that doing big changes in existing code to be challenging, especially while following an iterative and continuous process.

Product complexity Similar to continuous integration, the complexity of the SGSN-MME product has been identified as a challenge when breaking down software requirements. In particular, the high number of sub-systems makes coordination difficult and increases pressure according to one developer.

Another challenge is related to the main flow of the product, which should preferably be delivered together. This makes it hard to break it down into smaller units that can be worked on separately.

A configuration manager stated that the product is currently vulnerable to changes. As a result, multiple maintenance branches need to be maintained. Lowering the product's vulnerability to change would reduce the amount of maintenance work required as well as lessen customer inconvenience.

Large requirements Several interviewees (6 out of 13) report on requirements being too large. They can sometimes take up to 1000 man hours to implement, whereas the ideal is 200-300 man hours. Breaking down such requirements while maintaining delivery of customer value as originally intended is currently an issue. This means that there is a problem with finding the right balance when breaking down large software requirements. The ideal would be to have requirements that are not only easy to integrate and test (on their own) but also add value to the customer. Another factor that makes large requirement problematic is that they tend to change throughout development compared to smaller requirements due to their larger scope.

System tests for large requirements also tend to be too big, which slows down the integration process since large system tests take longer to run. This gets even more problematic when integration frequency is increased further due to long regression cycles. This is described further in section 5.1.2.

Low level requirements One developer identified as a challenge the breakdown of low level requirements to allow higher integration frequencies. For instance, requirements that involve low level algorithms that need to be tested as one unit could be hard to split up into several units. This is due to current quality assurance policies which do not allow integration of separate parts of a requirement. Doing so will result in failed test cases due to incomplete implementations.

5.2.2 Alignment of requirements and tests

Requirements need to be associated with tests for them to be properly integrated into the mainline, aligning them proved to be a challenge for the studied company.

Implementation and test dependencies Aligning the implementation of requirements and their tests is considered a problem by many developers. Previously, focus on testing was mainly at the end of a sprint. However, the introduction of continuous integration has shifted the focus of functional tests to each integration. One software developer argues that it is hard to sync requirements that need to be tested together when integrating.

Some requirements are simply too big so that they can be tested. Tests that are too big and too old. In fact, these requirements are big on purpose, otherwise they cannot be tested and delivered.

Another problem mentioned by one of the developers is synchronising test cases and their related product code on the integration track. It is often hard to know whether test cases can be integrated before the corresponding product code is available and vice-versa.

5.2.3 Customer value

Breaking down software requirements into small enough units to support more frequent integrations of code while keeping market priorities and customer value in mind can be tricky. The upcoming introduction of on demand software delivery (ODSD) further stresses the importance of this matter (see section 5.3.2). Moreover, this vision will require closer access to the customer.

Access to customer Since the introduction of continuous integration, collaborating with customer in regards to the requirements has been difficult according to some interviewees (4 out of 13). A CI driver mentioned that the teams are continuously trying to get closer to the customer. One developer stated that access to the customer differs from team to team.

The call for closer customer access for the teams comes after a change in the way requirements were handled. Previously, it was up to system managers to break down requirements since they had more knowledge. This task is now up to the product owner and the team. Although, access to the customer has improved with the product owner, many developers feel there is still more work to be done according to an agile coach. System managers had a good technical know-how but were not customer centric compared to the product owners who in turn have lower technical knowledge.

A balance between customer and technical perspectives is needed for breaking down software requirements, especially for the implementation of on demand software delivery (ODSD) later (see section 5.3.2).

Delivering customer value The studied company has introduced an additional requirement abstraction level, more specifically ”working scenarios”. This to allow requirements to be broken down into even smaller pieces thus enabling more frequent integrations. However, these working scenarios do not necessarily contain complete customer value.

One software developer argues that it does not make sense to deliver work at a ”working scenario” level, seen from a customer value perspective. Balancing the need of always delivering customer value and to breakdown software requirements into working scenarios that benefit the integration frequency is reported to be an issue.

Another software developer mentions the challenge of breaking down software requirements while keeping market priorities in mind. Additionally, one developer thinks that being scenario and test driven while breaking down software requirements to working scenarios could beneficial, however, this approach does not take delivery of customer value into account.

Finally, controlling whether or not you deliver feature growth and customer value at a healthy pace will be even more important when introducing on demand software delivery (ODSD), see section 5.3.2 for further information.

5.2.4 Guiding principle

A majority of the interviewees (11 out of 13) had a hard time describing the process used to break down software requirements at the studied company. Not having a clear unified process and the lack of guidance have been identified as possible reasons for this ambiguity. Additionally, the ongoing responsibility shift might complicate this matter even further (see 5.2.4).

Lack of guidance Some developers described a lack of guidance when it comes to breaking down software requirements. That is, how to break down requirements is left to the teams to decide. One developer thought this to be a good idea in the long run but not so much in the short term. Also, some believe that the issue of how to breaking down requirements has been missed since most people think that teams should just "practice continuous integration".

Some interviewees (5 out of 13) want more guidance from system managers who have better understanding of the technical aspects of some requirements. However, this is not always possible right now since the system manager's role has changed as one developer states:

"*The system manager actually produces an "early phases", and an "early phases" is a power point presentation describing the main part of the feature and that is handed over to the team and then they could do whatever they want but when they are ready they need to produce the code, test cases and the requirements in the ReqPro which is text based, and how they do that is up to the team.*" - Software Developer

The situation described by the developer in the above quote has led teams to develop their own approaches to handling requirements. Some have come farther than others. While these approaches are not in use in the entire SGSN-MME development program, they are nevertheless supported by management. The challenges identified in this sub-section are also related to the challenges presented in the next section.

No unified process Since the continuous integration pilot program started, teams were given a lot of freedom. They could set their own integration pace and decided how the requirements should be broken down. While this has been welcomed by most teams, especially more mature ones, some younger teams saw this as a lack of a unified process and made the transition more difficult for them.

Some developers think that there are no clear guidelines on how requirements break down ought to be done. This has led to a fragmented process, where different teams use different methods despite working towards the same goal. Results show that although pilot teams assist others in the process, the majority still rely on their knowledge of the product they are working on. As a result, some developers think that more coordination is needed in the future in order for a more unified process to be developed, as one developer puts it:

"*We're a big organisation, with quite a lot of teams and this coordination has not really been thought through how it's supposed to be done.*" - Software Developer

A line manager also mentioned that most teams learn by doing when it comes to breaking down requirements while support is provided by the pilot teams. However, improvements can be made with regards to the involvement of operational product owners and system managers.

The issue of finding a balance between the freedom given to the teams in relation to a bottom-up approach are presented in section 5.1.5.

Ongoing responsibility shift Some interviewees (4 out of 13) mention that the process and responsibility of breaking down software requirements is undergoing a total overhaul. Previously, this was carried out centrally by system managers. Now they are more integrated with the teams due to their contact with different stakeholders. However, it is still up to each team to carry out the breakdown of requirements. The pre-study phase carried out by system managers is more of a cost input to the management. The process overhaul has introduced more communication responsibilities with the customer for each team which has been challenging due to lack of experience. One software developer believes that the process overhaul, though a change for the better in the long run, could be an obstacle to the implementation of continuous integration in the short term. The situation is currently described as pretty chaotic by one software developer:

"*...everything is pretty chaotic I would say,... before you always got clear requirements,... Right now we get mails with like three sentences and everything is very fluffy... you need a lot of feedback always from system management, system architects and so on, to find out what they really want...*" - Software Developer

One developer mentions that access to the customer plays an important role in the challenges presented above (see 5.2.3). When more access to the customer is possible, the current ongoing change will aid the process of breaking down software requirements and also the adoption of continuous integration.

Some interviewees (4 out of 13) problematise the process overhaul due to the lack of experience in breaking down software requirements and customer collaboration within teams. For instance, one line manager is of the opinion that system managers should be more involved in the introduction of continuous integration to support the teams. Especially now when the task of breaking down software requirements has been overhauled.

Unfit process Results from this study indicate that the process used by the teams is not mature enough and could be improved.

One suggestion, that has been presented by a developer, is that requirements analysis should be done at the team level instead of at the system manager level (this change is ongoing). Moreover, teams should produce a prioritized sprint backlog (with features that have been broken down into reasonable size) along with a sprint proposal for the system managers. These steps should be done one sprint at a time. This suggestion would allow more feedback from the teams during sprint planning, which currently is described as a challenge.

Interestingly, the situation is slightly different in Shanghai, where a CI driver mentions that teams tend to move between different subsystems, hence the lack of knowledge while breaking down software requirements is currently an issue.

One developer would like to have more involvement from operational product owners (OPO) during the process of breaking down software requirements, however, this issue is currently being addressed, as a step towards becoming more Agile.

5.2.5 Summary

Several challenges that may arise while breaking down software requirements have been presented in this section. Some within the context of continuous integration whilst others more general to the concept. The presented challenges include *requirements abstraction*, *guiding principle*, *customer value* and *alignment of requirements and tests.*

5.3 RQ3 - Software requirements breakdown's influence on continuous integration

This section answers the third research question of this study, how requirements breakdown influence the adoption of continuous integration. The section starts by introducing the necessity of breaking down requirements within continuous integration followed by the implications.

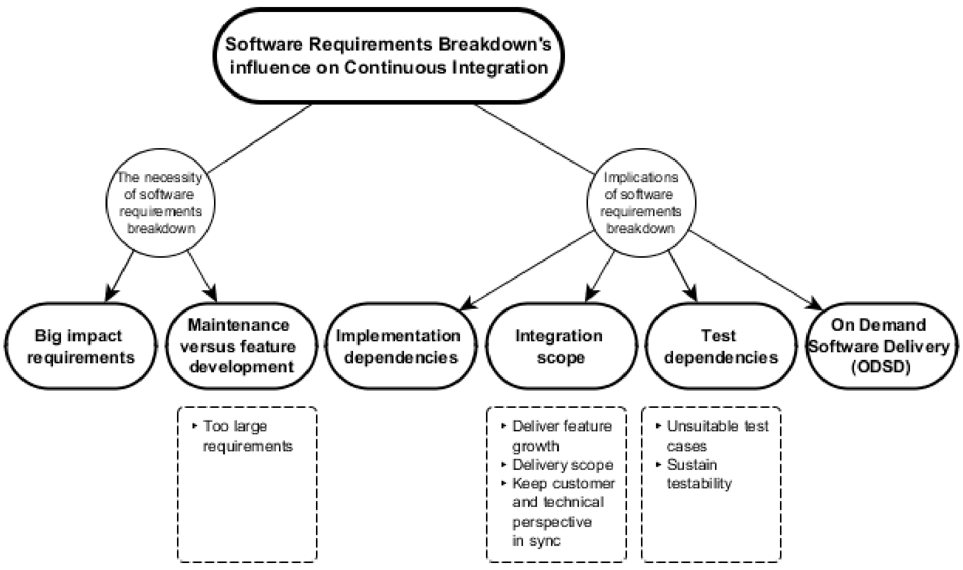

Software Requirements Breakdown's influence on Continuous Integration
The necessity of software requirements breakdown
Implications of software requirements breakdown
Big impact requirements
Maintenance versus feature development
Implementation dependencies
Integration scope
Test dependencies
On Demand Software Delivery (ODSD)
Too large requirements
Deliver feature growth
Delivery scope
Keep customer and technical perspective in sync
Unsuitable test cases
Sustain testability

Figure 5.3: Thematic map of software requirements breakdown's influence on continuous integration

The aggregated source count for each theme is as follows: Big impact requirements: 2, Maintenance versus feature development: 2, Implementation dependencies: 3, Integration scope: 4, Test dependencies: 2 and On demand software delivery (ODSD): 4. For further detail regarding source and reference count for each theme and sub-theme see Appendix B.

5.3.1 The necessity of software requirements breakdown

Results from this study indicate that there is a difference in opinion whether or not you need to break down software requirements in order to adopt a efficient continuous integration process with an integration frequency that is beneficial. For instance, one agile coach is of the opinion that feature breakdown is a "nice to have" thing rather than a necessity:

"*Break down the features into user stories is necessary for Agile and Scrum, but for continuous integration breakdown the user story to scenario and then you deliver the scenario to the main branch is a 'nice to have' -thing.*" - Agile Coach

In addition, the same agile coach argues that breakdown of user stories to working scenarios should not be seen as mandatory for continuous integration:

"*Break down the user story into scenario is not a mandatory thing for us to do the CI - that's my opinion about this.*" - Agile Coach

On the contrary, one line manager is of the opinion that software requirements breakdown is a prerequisite to the adoption of continuous integration. This is because partial and untestable code cannot be integrated. Therefore, it is important to properly break down software requirements to facilitate their integration later.

One software developer mentions that the need of breaking down software requirements depends on the required integration frequency. Moreover, more frequent integrations allow teams to know what other teams are currently working on which could ease the breakdown and specification of those teams' requirements.

Maintenance versus feature development Teams that are developing new features have been facing more difficulties while transitioning to continuous integration compared to teams that are mainly maintaining existing code and fixing bugs.

One CI driver who is also leader of a maintenance team, mainly working with correcting bugs, argues that the adoption of continuous integration has been much smoother for them. Maintenance tasks (e.g. bug fixing) undertaken by this team, do not necessarily need to be broken down to allow unitary integration. The CI driver is convinced that the fact that they do not need to break down software requirements, due to the nature of their tasks, is the reason for their smooth adoption of continuous integration.

Additionally, several interviewees mention that it is the one to one mapping of requirements, test cases and integrations that complicates the matter for feature development teams. By allowing integration of partial functionality, test cases, working scenarios or similar might benefit the breakdown of software requirements with regards to the technical implementation perspective. Less restrictions on the integration line would put less focus on the need to break down software requirements with regards to high integration frequency.

Features development teams often deal with issues of *too large requirements* while adopting continuous integration. This has forced system managers to come up with smaller requirements making the task of breaking them down easier for each team. Moreover, system tests for large requirements take longer to run which is not in accordance with continuous integration principles.

Big impact requirements As mentioned above, teams that are mainly involved in product feature development have more difficulties adopting continuous integration due to the issues in breaking down requirements.

Breaking down software requirements that have a major impact on the system or that are considered high risk have resulted in the use of two separate tracks for development. One part is done using continuous integration while the other is implemented in a more ”traditional” way. For instance, CPU load is an important non functional requirement for SGSN-MME customers. Hence, any requirement that could potentially affect CPU load is deemed a high risk requirement that has a major impact on the system. Such a requirement requires a lot of testing that is often manual. The tests themselves are also old and unstable. For these reasons, big impact requirements are being developed without continuous integration and thus integrated less frequently.

5.3.2 Implications of software requirements breakdown

The need of breaking down software requirements seems more widespread among feature development teams as described above. Results show that breaking down software requirements while using continuous integration entails certain implications which are presented below.

Implementation dependencies Finding the right balance between small enough requirement units and high integration frequency without compromising the implementation unity is reported to be an issue at the studied company. That is since continuous integration promotes high integration frequencies, software requirements might need to be split up accordingly. However, dependencies within the implementation of said requirement might benefit from being developed as a whole unit. Moreover, implementation dependencies might not be identified before the actual development is carried out, as described by a CI driver:

"*I heard one example from the feature development team. They had to break a user story into 3 scenarios. They are trying to implement the 3 scenarios separately. However, later on during the implementation they found dependencies between the 3 scenarios so they have to combine the 3 scenarios later together. It's like they separate 3 work branches to work on the individual scenarios. However, they have to merge them together.*" - CI Driver

This quote, as illustrated in figure 5.4, highlights the struggle that might arise when requirements have been broken down according to the integration frequency: each working scenario is first implemented on its own branch, but later merged back to one working branch due to identified implementation dependencies. This split and merge of branches takes valuable time from development.

As stated by an agile coach and a CI driver, currently, breaking down software requirements into smaller units increases the need of multiple working branches. This increases the need of maintenance and complicates the matter of implementation dependencies between branches. Although, if the current branch strategy would allow integration of work that does not cover complete functionality with test cases the number of branches needed to maintain could be decreased.

Test dependencies While breaking down software requirements into smaller units, it was found that test dependencies played an important role. These smaller tasks need to be individually testable if they are to be integrated separately. At the studied company, this was often not possible because of *unsuitable tests*: many are old and large in nature. This means that smaller work units cannot be tested separately. Instead, a complete feature needs to be integrated together and only then can it be tested in its entirety. Consequently, *sustaining testability* of smaller work units was found to be an important implication for breaking down software requirements.

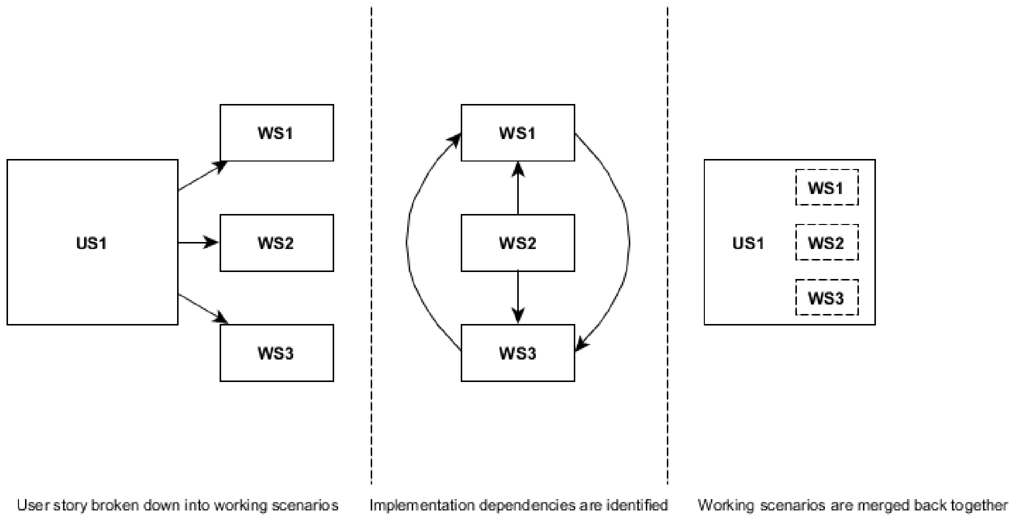

US1
WS1
WS2
WS3
WS1
WS2
WS3
US1
WS1
WS2
WS3
User story broken down into working scenarios
Implementation dependencies are identified
Working scenarios are merged back together

Figure 5.4: Illustration of how implementation dependency issues might arise from breaking down a user story.

On Demand Software Delivery (ODSD) Ericsson's vision in the next couple of years is to implement On Demand Software Delivery (ODSD). While the connection of continuous integration to ODSD is not clear. The findings of this study show that continuous integration has at least a few implications on transitioning to ODSD. One line manager is of the the opinion that continuous integration will facilitate the migration to ODSD but that the frequency of integrations needs to be determined first. Also, the mapping of tests and requirements is equally important: according to a CI driver, this mapping is necessary for ODSD since customers will be able to verify a requirement in relation to its tests. This would lead to increased trust and minimise the risk for misunderstandings. Similarly, it seems that proper and careful break down of requirement is necessary when delivering software on demand to the customer. One developer states that in the case of SGSN-MME product, if 30 teams are working and want to deliver new functionality to the customer then it is important to have control over what is delivered, supported, which feature works (or parts of it) and so on.

Integration scope One implication of breaking down software requirements and integrating them in smaller chunks is that partial code might be integrated into the Pre-Test Build (PTB). This could potentially, cause exceptional behaviour to the product according to a line manager. As a result, defining the right *integration scope* is important.

Related to the integration scope is the *delivery of feature growth* when breaking down software requirements. Results suggest that integrating code should provide new value to the customer and contribute feature growth. This is however not always the case at the studied company (see 5.1.7).

Synchronising both the customer and technical perspective is also of significance when it comes to the integration scope. Often times, the customer wants parts of the product delivered in a certain order for different reasons. Honoring that order from a technical perspective is something that the studied company needs to address. Especially in regards to On Demand Software Delivery (introduced above).

5.3.3 Summary

There is a difference in opinion regarding the necessity of breaking down software requirements while adopting continuous integration. There seems to be a difference in the ease of transitioning towards continuous integration depending on the work being carried out. Yet, breaking down software requirements into small enough units to support more frequent integrations of code might come with certain implications.

6

Discussion

This section discusses the results presented in section 5 and ties them to earlier research literature. Section 6.1 focuses on discussing the first research question - the challenges of continuous integration. Section 6.2 discusses the challenges of requirements break down (RQ2). The section ends with a discussion of the third research question in 6.3.

6.1 RQ1 - Continuous integration adoption challenges

The findings indicate that adopting the mindset aligned with continuous integration is a challenge. Interviewees were skeptical about the benefits that they could gain from adopting continuous integration. Similarly, Claps [7] found that teams adopting continuous deployment need to adopt the mindset needed for it. This means that there needs to be a shift towards a single and united organisational culture that adopts the principles of continuous integration. The studied company is transitioning to continuous integration in order to be able to integrate more often and deliver better quality software according to the interviewees. This implies that adopting continuous integration can be seen as introducing a change to an existing software process. Also, bringing change to an existing process in an organisation with the aim of improving software quality is considered a software process improvement endeavor [9]. As such, continuous integration is a software process improvement initiative undertaken by the studied company in a bid to further increase the efficiency of the software development process. Indeed, The SGSN-MME program is transitioning to continuous integration in order to improve integration frequency and quality. That said, Mathiassen et al. [28] argue that improving a process (such as implementing continuous integration) involves changing people. They argue that people do not change simply because processes change. As such resistance to change should be expected. This could explain why some developers express some degree of scepticism towards the adoption of continuous integration. Another reason could be the lack of motivation to change, due to the complexity of adopting a new process, especially if the change is perceived as being too complex [8].

Changing work habits has also been problematic. People were found to be afraid to show their work through early and frequent integrations. This fear could stem from the transparency that continuous integration advocates. Indeed, Conboy et al [8] found that developers at 17 different companies adopting agile were afraid of exposing their work intentions since it could potentially expose their weaknesses. The authors [8] stress the importance of providing a safe environment where mistakes and flaws are not judged.

Results show that all of the 13 interviewees mentioned challenges related to tools and infrastructure such as code review, regression feedback time when transitioning to continuous integration. The maturity of the tools and infrastructure was found to be a major issue. This has been mentioned in earlier literature. For instance, Olsson et al. [32] identified tools with support for automated tests as a barrier when adopting continuous integration in two companies. Similarly, they argue that developing a fully automated infrastructure remains the key focus when adopting continuous integration [32]. At the SGSN-MME program, the tools and the infrastructure are not entirely ready to accommodate continuous integration. Many developers feel that the current tools available are holding them back. This means that teams are still not able to fully exploit the benefits of continuous integrations. That said, a lot of progress is being made such as developing new tools to better support automated tests and the reduction in the amount of branches which is a challenge in itself [32]. The maturity issue is most likely due to the fact that continuous integration is a new concept that is still being adopted. Also, the ongoing responsibility shift that the SGSN-MME program is going through (see section 5.2.4) means that tools and infrastructure need more time to adapt.

Another issue worth discussing are the tests themselves. The need for automated tests when transitioning to continuous integration has been recognized in previous research [14, 20]. While organisations realise the importance of automated tests when moving towards continuous integration, they still struggle with it [32]. Improving the testing tools and the infrastructure is not enough according to the interviewees. Olsson et al [32] have also identified automated tests as a key barrier when transitioning from agile to continuous integration. In order to mitigate this problem, two companies facing this issue, have made it their priority to greatly increase the number of automated tests [32] since this is important for continuous integration [14]. Doing this at Ericsson might not be a trivial task due to the nature of the requirements they are facing. Some are too large and cannot be tested separately (at least for now). Therefore, writing automated tests might not currently be an option. For instance, one issue mentioned by some of the developers is automating the tests for some features that are part of the Linux kernel used in the product. This issue is similar to the problem encountered by the developers of the FreeBSD project when they were implementing the Symmetric Multi-Processing (SMP) module in the Linux kernel using continuous integration [19]. The ongoing struggle with manual tests could be due to to the complexity of the product itself. SGSN-MME has been developed for well over a decade, updating the old and large test cases could be a tedious and time consuming task.

Results from this research show that there is an ambiguity regarding the goals and the organisational vision for the implementation of continuous integration. More than half (7 of 13) of the interviewed subjects had a hard time answering questions due to their lack of knowledge of what was expected from them personally. According to Kotter [24], successful cases of process change all share a common denominator liable for their success of change implementation. This denominator is divided into a series of 8 phases [24]: "establishing a sense of urgency", "forming a powerful guiding coalition", "creating a vision", "communicating the vision", "empowering others to act the vision", "planning for and creating short-term wins", "consolidating improvements and producing still more change" and "institutionalizing new approaches". Kotter [24] highlights the importance of spending the necessary amount of time on each step, and failing to do so will lead to undesired results. For instance, lacking and under-communicating a vision might lead to confusion and incompatible results, which will take the organisation in the wrong direction. This might be a reason to the unclear goals and expectations regarding continuous integration at the studied company. Another reason could be the use of a bottom-up approach where directions and guidance come from experience gained in pilot teams. This might have led to the confusion regarding a vision, since the most expected communication channel for organisational visions ought to be management. However, this is according to the authors own opinion. According to research done by Ståhl and Bosch [46] there is no general consensus regarding the practice of continuous integration. Besides the fact that this makes the comparison between different continuous integration practitioners difficult, it could be a reason for the ambiguity at the studied company as well.

The bottom up approach seems to suit Ericsson's needs and goals related to adopting continuous integration. The alternative is a top down approach which assumes that there is a universal set of practices that could be adopted by all software organisations related to the change being implemented [48]. Similarly, Zucconi [53] states that the top down approach promotes a "one size fits all" manner when it comes to implementing process changes in a software organisation. A bottom up approach on the other hand emphasises that process improvements be guided by an organisation's own objectives, products and challenges [48]. This seems to be in line with the situation at the studied company, several interviewees spoke of the need to adopt a well paced continuous integration process that it is suitable to Ericsson and its goals, size and products. This seems to be the right approach to take and most teams are happy with having the ability to find their own pace when it comes to adopting continuous integration as suggested by a CI driver. Some developers believe that most continuous integration practices outlined in different guides and books are more suitable for Internet based products and websites.

Findings from this study show that the suitability of continuous integration is questioned by the majority of subjects interviewed (11 of 13 interviewees). Interviewees mentioned the complexity of the product and the industry in which the studied company operates as not being ideal for continuous integration. Research done by Olsson et al [32] describes barriers identified in hardware oriented companies moving towards continuous integration. These companies are used to hardware oriented processes and are struggling with adopting a software oriented process such as continuous integration. Therefore a shift in culture is needed. Additionally, research done by Turk et al. [49] on the suitability of agile development methods shows that assumptions (e.g. face-to-face communication, quality assurance, changing requirements) made by such methods are not appropriate for all organisations, products and projects. The authors [49] highlight important limitations of said methods, where two is of particular interest for this research, namely limited support for large complex software and large teams. Agile development methods are considered a prerequisite for continuous integration [32], therefore findings by Turk et al. [49], might apply to the applicability of continuous integration. Therefore, the suitability issues of continuous integration raised by interviewees are most likely due to the complexity of the product and the number of people working on it.

6.2 RQ2 - Software requirements breakdown challenges

Requirements abstraction has been found to be problematic for many teams. The interviewees have described the requirements as ambiguous when they are about to be implemented. Meaning that requirements are on the wrong level of abstraction when they are handed over to designers and developers for implementation. As mentioned in section 5.2.1, developers sometimes get a few sentences that describe a requirement and are from that supposed to return with a prototype. However, it seems that such a short description for a requirement does not guarantee a proper implementation. Gorschek and Wohlin [17] mention that for a requirement to be implemented, it must be testable and unambiguous and be on the component or function level according to the Requirements Abstraction Model (RAM) [17]. It seems that this is not always the case here.

Besides being ambiguous, requirements are also too large for many developers that are using continuous integration. This contributes to the problem of requirements abstraction. These requirements require more than 1000 man hours. Results show that teams are aware of this issue and its negative impact on their ability to integrate more often. The problem of large requirements and the need to break them down into smaller tasks for continuous integration seems to introduce another issue: the balance of being able to deliver small parts of a requirement and delivering customer value. Some developers believe that they must be allowed to deliver smaller parts of a feature while others feel some parts are too small or insignificant and do not warrant a separate integration. The complexity of the SGSN-MME product as well as its architectural design also increase the complexity of dependencies between the requirements and their impact on different subsystems. The complexity of managing requirements is also related to the inter-dependencies among requirements [5] which has been reported has a challenge by developers. Furthermore, the telecoms market in which Ericsson operates could also contribute to the problem of abstraction. The telecoms industry is a market driven one as defined by Gorschek and Wohlin [17], requirements are elicited from several sources which means they come in different levels of abstraction. This difference could explain why some requirements are on an unsuitable level of abstraction when they reach the developers.

Related to breaking down software requirements is delivering customer value. This issue has been mentioned by several interviewees. Adopting continuous integration will eventually result in a higher number of integrations but with smaller scope (per integration). However, some integrations might not deliver visible customer value suggests a line manager. It seems that how to determine the right scope and frequency of integrations has not been pinned down yet. Also related to this is aligning the requirements and tests. One developer mentioned that customers often want to see which tests a new feature/change has passed. This is hard at the moment because of the aforementioned barriers related to tools and testing when using continuous integration.

The SGSN-MME program has no formal process of breaking down requirements. Similar to continuous integration, a bottom up approach is employed where each team is free to adopt their own method at their own pace. This has however, resulted in some teams being more successful than others when breaking down software requirements. There is a team that acts as a role model and provides support which seems to be appreciated by other teams. However, the lack of a guiding principle could be due to a ongoing responsibility shift. That said, the ongoing responsibility shift alongside the simultaneous introduction of continuous integration could be a reason behind the challenges associated with the adoption of continuous integration and the breakdown of software requirements.

6.3 RQ3 - Software requirements breakdown's influence on continuous integration

Whether or not the breakdown of software requirements should be considered mandatory for the adoption of continuous integration has been discussed back and forth throughout the interviews. One agile coach is of the opinion that breaking down software requirements should be considered a 'nice to have' -thing rather than mandatory or a prerequisite for continuous integration while one developer thinks that it depends on the current integration frequency. That is, higher integration frequencies require smaller requirement units. On the contrary, one line manager argues that it should be considered a prerequisite, this to properly facilitate the integration of code and corresponding test cases. Interestingly, results from this study show that a maintenance team, mainly working with correcting bugs, has had a much smoother adoption of continuous integration (see section 5.3.1). That is, mostly due to the fact that this maintenance team receives bugs that can each be easily worked on, tested and integrated unitarily. This seems to be what the other teams that are working with feature development are trying to achieve by breaking down software requirements. Research done by Holck and Jorgensen [19] mention that two open source projects using continuously integration actually break down work to allow higher integration frequencies. Additionally, Olsson et al. [32] identified long lead times, due to comprehensive build processes, as problematic while adopting continuous integration. According to Olsson et al. [32], this could be addressed by modularizing development into smaller units. These two examples from previous research might suggest that breaking down software requirements into smaller work units is needed or at least recommended. One reason behind the difference in opinion regarding the necessity of breaking down software requirements at the studied company might be that teams deal with different types of work (e.g. maintenance vs new feature development). Still, it seems that this mostly proves the need of breaking down software requirements rather than the difference in opinion. Another reason might be the difference in integration frequency between teams. Teams that have achieved a higher integration frequency might feel the need of breaking down software requirements even further, as opposed to teams with a lower integration frequency. That said, the need of breaking down software requirements in order to adopt continuous integration is most likely dependent on the sought after integration frequency.

Breaking down software requirements has been found to introduce implications for teams at the studied company. For instance, one CI driver mentions that implications related to implementation dependencies may arise when requirements that otherwise might benefit from being developed as a whole unit are broken down into several units. Integrating partial code into the Pre-Test Build (PTB) could potentially cause exceptional behaviour to the product according to a line manager. In addition, breaking down requirements solely with the technical perspective in mind might introduce implications related to the delivery of customer value. Research by Holck and Jorgensen [19] reports on implications that occurred in one open source project taking on a very large feature using continuous integration. At first, the feature was organized into a sub-project, however, to avoid a 'big bang integration' later on, they decided to perform the integration incrementally. However, doing so would later prove to bring significant challenges while trying to keep the development version in a working state. The following quote [19, p. 46]: "transforming a van into a sports car while driving", was used to describe the task. One reason behind the implications found at the studied company might be due to the strict quality assurance policy of the product branch. That said, allowing integration of work that does not cover complete functionality with test cases could most likely solve most of the implications.

7 Conclusion

This study set out to identify the challenges of continuous integration and requirements break down and how the latter influences the implementation of a continuous integration process. Through a single case study, 13 semi-structured interviews were conducted and a set of challenges related to the adoption of continuous integration (see section 5.1) and software requirements breakdown (see section 5.2) were identified.

In relation to RQ1 - continuous integration adoption challenges, the most dominant results include: 1) The mindset is an important factor in the success of implementing continuous integration. Scepticism towards the introduction of a new process needs be considered in order to win over non believers. 2) Testing tools and the maturity of the infrastructure supporting the continuous integration process is required in order to facilitate the daily tasks involved. Continuous integration advocates the use of automated tools to allow more frequent and efficient integrations. 3) Similar to Agile, assumptions made by the concept of continuous integration may not apply to all organisations, products and projects, especially those of larger dimensions. 4) Lacking and under-communicating a clear goal can delay and cause confusion among developers when transitioning to continuous integration.

Some of the identified challenges such as mindset, tools and infrastructure maturity and testing have been mentioned in previous literature when transitioning to continuous integration. However, this study also identified software requirements as a challenge when adopting continuous integration. There is currently a lack of research in this regard, to the authors best knowledge.

The findings regarding RQ2 - software requirements breakdown challenges, the most prominent results are: 1) Requirements abstraction plays a deciding factor in the success of breaking down software requirements. Requirements that are too large or ambiguous complicates the matter of finding the right level of abstraction of a requirement. 2) Breaking down software requirements to allow more frequent integrations makes it hard to guarantee delivery of customer value since some tasks or units of requirements might not seem like a benefit to the customer. That said, keeping both the customer and the implementation perspective in mind while breaking down software requirements is important. 3) Providing a clear guiding principle is important in order to facilitate the break down of requirements for the teams.

Though the focus of this research question is on the challenges of software requirements breakdown, they are inherently contextualised by the adoption of continuous integration.

When it comes to RQ3 - software requirements breakdown's influence on continuous integration, the majority of the discussion includes: 1) The need of breaking down software requirements in order to allow more frequent code integrations correlates with the sought after integration frequency. 2) Breaking down software requirements implies finding a balanced integration scope as well as considering test and implementation dependencies. However, keeping a flexible quality assurance policy of the product branch might increase the possibilities of breaking down software requirements with regards to continuous integration.

In conclusion, it seems that the role of software requirements and their breakdown plays an important role when adopting continuous integration, especially with regards to an increased integration frequency.

7.1 Implications for practitioners

This study presents a set of challenges of continuous integration and breaking down software requirements. These challenges can be used as a checklist by companies that are about to transition to continuous integration. Also, this thesis gives software organisations insight into the roles of software requirements and their breakdown as well as the related challenges when adopting continuous integration.

Practitioners about to transition to continuous integration may use results from this research as a decision making tool. It allows them to compare their situation with the findings in order to avoid some of the barriers that may arise.

7.2 Contributions to academic research

This study adds to the existing body of academic knowledge within software engineering. Not only does this thesis deal with the lack of studies within continuous integration as a research field but also the lack of studies related to software requirements break down.

As opposed to prior research on continuous integration, this study adds a more focused description of the challenges that organisations face when adopting continuous integration and breaking down software requirements. These challenges are presented in section 5.1 and section 5.2. This insight has not received much attention in prior research, according to the researchers best knowledge.

Additionally, the influence software requirements breakdown has on the adoption of continuous integration has been investigated and are presented in more detail in section 5.3. Though previous research has touched upon the subject, this research discusses the need of breaking down software requirements in order to adopt a continuous integration process in more detail. Also, implications that the breakdown of software requirements might introduce have been investigated.

7.3 Future work

This case study focused on studying a single company. It would be interesting to involve multiple companies. This in order to see if the results found are unique to the studied company or could apply to other companies as well. Also, further studies on how software requirements break down affects continuous integration should be conducted. To the authors best knowledge, little research has been done in this area. It would also be beneficial to further investigate how the challenges identified in this study can be avoided and potentially develop a set of guidelines that can be tested and validated in industry. Finally, looking at results from this research it would be interesting to further investigate the relationship between continuous integration and testing as well as infrastructure maturity.

Bibliography

[1] Berntsson Svensson, R.; Olsson, T., , Regnell, B. Introducing support for release planning of quality requirements - an industrial evaluation of the quper model. In *Software Product Management, 2008. IWSPM '08. Second International Workshop on*, pp. 18–26, Sept 2008.

[2] Berntsson Svensson, R.; Gorschek, T.; Regnell, B.; Torkar, R.; Shahrokni, A.; Feldt, R., , Aurum, A. Prioritization of quality requirements: State of practice in eleven companies. In *RE*, pp. 69–78. IEEE, 2011.

[3] Boegh, J. A new standard for quality requirements. *Software, IEEE*, 25(2):57–63, March 2008.

[4] Braun, V. , Clarke, V. Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3(2):77–101, 2006.

[5] Carlshamre, P.; Sandahl, K.; Lindvall, M.; Regnell, B.; Natt, , , Dag, J. An industrial survey of requirements interdependencies in software product release planning. In *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*, pp. 84–91, 2001.

[6] Ceschi, M.; Sillitti, A.; Succi, G., , De Panfilis, S. Project management in plan-based and agile companies. *Software, IEEE*, 22(3):21–27, May 2005.

[7] Claps, G. Continuous deployment: An examination of the technical and social adoption challenges. diploma thesis, The University of New South Wales, 2012.

[8] Conboy, K.; Coyle, S.; W., Xiaofeng, , Pikkarainen, M. People over process: Key challenges in agile development. *Software, IEEE*, 28(4):48–57, July 2011.

[9] Deependra, M. Managing change for software process improvement initiatives: a practical experience-based approach. 4(4):199–207, 1998.

[10] Dybå, T. , Dingsøyr, T. Empirical studies of agile software development: A systematic review. *Information & Software Technology*, 50(9-10):833–859, 2008.

[11] Ericsson, AB. Ericsson SGSN-MME @ONLINE. URL `http://www.ericsson.com/ourportfolio/products/sgsn-mme`.

[12] Ericsson, AB. This is Ericsson @ONLINE, 2012. URL `http://www.ericsson.com/res/thecompany/docs/this-is-ericsson.pdf`.

[13] Fitz, T. Continuous deployment at imvu: Doing the impossible fifty times a day @ONLINE, 2009. URL `http://timothyfitz.com/2009/02/10/continuous-deployment-at-imvu-doing-the-impossible-fifty-times-a-day/`.

[14] Fowler, M. Continuous integration @ONLINE, May 2006. URL `http://martinfowler.com/articles/continuousIntegration.html`.

[15] Gilb, T. *Competitive engineering.* Elsevier Butterworth Heinemann, 2006.

[16] Goodman, D. , Elbaz, M. ”it's not the pants, it's the people in the pants” learnings from the gap agile transformation what worked, how we did it, and what still puzzles us. In *Agile, 2008. AGILE '08. Conference*, pp. 112–115, Aug 2008.

[17] Gorschek, T. , Wohlin, C. Requirements abstraction model. *Requirements Engineering*, 11(1):79–101, 2006.

[18] Harald, S. , Höst, M. Views from an organization on how agile development affects its collaboration with a software development team. In Bomarius, F. , Komi-Sirvia, S., editors, *Product Focused Software Process Improvement*, volume 3547 of *Lecture Notes in Computer Science*, pp. 487–501. Springer Berlin Heidelberg, 2005.

[19] Holck, J. , Jørgensen, N. Continuous integration and quality assurance: a case study of two open source projects. *Australasian J. of Inf. Systems*, 11(1), 2003.

[20] Humble, J. , Farley, D. *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation.* Addison-Wesley Professional, 1st edition, 2010.

[21] Hüttermann, M. *DevOps for Developers.* Apress, Berkely, CA, USA, 1st edition, 2012.

[22] Karlsson, J. , Ryan, K. A cost-value approach for prioritizing requirements. *Software, IEEE*, 14(5):67–74, Sep 1997.

[23] Klein, H.K. , Myers, M.D. A set of principles for conducting and evaluating interpretive field studies in information systems. *MIS Quarterly*, 23(1):67–93, 1999.

[24] Kotter, John P. Leading change: Why transformation efforts fail. *Harvard Business Review*, 85(1):96, Jan 2007.

[25] Lacoste, F.J. Killing the gatekeeper: Introducing a continuous integration system. In *Agile Conference, 2009. AGILE '09.*, pp. 387–392, 2009.

[26] Mann, C. , Maurer, F. A case study on the impact of scrum on overtime and customer satisfaction. In *Agile Conference, 2005. Proceedings*, pp. 70–79, July 2005.

[27] Mannaro, K.; Melis, M., , Marchesi, M. Empirical analysis on the satisfaction of it employees comparing xp practices with other software development methodologies. In *Extreme Programming and Agile Processes in Software Engineering*, volume 3092 of *Lecture Notes in Computer Science*, pp. 166–174. Springer Berlin Heidelberg, 2004.

[28] Mathiassen, L.; Ngwenyama, O.K., , Aaen, I. Managing change in software process improvement. *Software, IEEE*, 22(6):84–91, Nov 2005.

[29] Miles, M.B. , Huberman, A.M. *Qualitative Data Analysis: An Expanded Sourcebook.* Sage, 2 edition, 1994.

[30] Miller, A. A hundred days of continuous integration. In *Agile, 2008. AGILE '08. Conference*, pp. 289–293, Aug 2008.

[31] Miller, A. , Carter, E. Agility and the inconceivably large. In *Agile Conference (AGILE), 2007*, pp. 304–308, Aug 2007.

[32] Olsson, H.H.; Alahyari, H., , Bosch, J. Climbing the "stairway to heaven" – a multiple-case study exploring barriers in the transition from agile development towards continuous deployment of software. pp. 392–399, Sept 2012.

[33] Paasivaara, M.; Lassenius, C.; Heikkila, V.T.; Dikert, K., , Engblom, C. Integrating global sites into the lean and agile transformation at ericsson. In *Global Software Engineering (ICGSE), 2013 IEEE 8th International Conference on*, pp. 134–143, Aug 2013.

[34] Pikkarainen, M.; Salo, O.; Kuusela, R., , Abrahamsson, P. Strengths and barriers behind the successful agile deployment - insights from the three software intensive companies in finland. *Empirical Software Engineering*, 17(6):675–702, 2012.

[35] Poppendieck, M. , Cusumano, M.A. Lean software development: A tutorial. *Software, IEEE*, 29(5):26–32, Sept 2012.

[36] Poppendieck, T. , Poppendieck, M. *Lean Software Development: An Agile Toolkit.* Addison-Wesley Professional, 1 edition.

[37] Regnell, B.; Berntsson Svensson, R., , Wnuk, K. Can we beat the complexity of very large-scale requirements engineering? In *Requirements Engineering: Foundation for Software Quality*, volume 5025, pp. 123–128. Springer Berlin Heidelberg, 2008.

[38] Ries, E. Continuous deployment in 5 easy steps @ONLINE, 2009. URL `http://radar.oreilly.com/2009/03/continuous-deployment-5-eas.html`.

[39] Robson, C. *Real World Research: A Resource for Social Scientists and Practitioner-Researchers.* Regional Surveys of the World Series. Wiley, 2002.

[40] Runeson, P. , Höst, M. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, April 2009.

[41] Schumacher, J. Continuous deployment at atlassian @ONLINE, 2011. URL `http://blogs.atlassian.com/2011/02/continuous_deployment_at_atlassian/`.

[42] Seaman, C.B. Qualitative methods in empirical studies of software engineering. *Software Engineering, IEEE Transactions on*, 25(4):557–572, Jul 1999.

[43] Sharp, H.; Finkelstein, A., , Galal, G. Stakeholder identification in the requirements engineering process. In *Database and Expert Systems Applications, 1999. Proceedings. Tenth International Workshop on*, pp. 387–391, 1999.

[44] Shenton, A.K. Strategies for ensuring trustworthiness in qualitative research projects. *Education for Information*, 22(2):63–75, 2004.

[45] Shenton, A.K. , Hayter, S. Strategies for gaining access to organisations and informants in qualitative studies. *Education for Information*, 22(3-4):223–231, 2004.

[46] Ståhl, D. , Bosch, J. Modeling continuous integration practice differences in industry software development. *J. Syst. Softw.*, 87:48–59, January 2014.

[47] Svensson, H. , Höst, M. Introducing an agile process in a software maintenance and evolution organization. In *European Conference on Software Maintenance and Reengineering*, pp. 256–264, March 2005.

[48] Thomas, M. , McGarry, F. Top-down vs. bottom-up process improvement. *Software, IEEE*, 11(4):12–13, Jul 1994.

[49] Turk, D.; France, R., , Rumpe, B. Assumptions underlying agile software-development processes. *Journal of Database Management*, 16(4):62–87, Oct 2005.

[50] Van Der Storm, T. Backtracking incremental continuous integration. In *Software Maintenance and Reengineering, 2008. CSMR 2008. 12th European Conference on*, pp. 233–242, 2008.

[51] Wilson, F. Continuous deployment @ONLINE, 2011. URL `http://www.avc.com/a_vc/2011/02/continuous-deployment.html`.

[52] Yin, Robert K. *Case study research: Design and methods*. Sage, 1994.

[53] Zucconi, L. Software process improvement paradigms for it industry: Why the bottom-up approach fits best. In *Software Engineering Conference, 1995. Proceedings., 1995 Asia Pacific*, pp. 511–, Dec 1995.

A

Interview Questions

Interview Questions	**Category**
1. What is your current role in your organisation? 2. How many years have you been working for your current organisation? 3. Which product(s) are you currently working on? 4. For how long have you been doing work related to software development? 5. Could you briefly describe which agile software development methodologies your current team has used?	Background

6. Are you familiar with the concept of Continuous Integration? • IF YES: 7. How would you describe what Continuous Integration is? • IF NO: 8. Just from hearing the phrase, what do you think Continuous Integration might mean? • Provide definition of CI: *"Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible"* [14] 9. Does your team currently use Continuous Integration to integrate software? • IF YES: 10. How long has continuous integration been used within your team? 11. Approximately how often do **you personally** integrate work using continuous integration? 12. Approximately how often does **your** team integrate using continuous integration? 13. Approximately how often did your team integrate their work before using continuous integration? • IF NO: 14. Approximately how often does your team integrate their work?	Understanding
RQ1: What are the challenges of implementing a continuous integration process in practice?	

• *Mention the beginning of continuous integration section* • *Ask follow-up questions to refine the granularity and detail of responses* 15. Are there any challenges that you or your team have faced or believe can potentially face when using Continuous Integration?	Open-ended question
16. Do you feel that you and/or your work colleagues should use Continuous Integration? (why/why not)	Individual Motivation
17. How well does the Continuous Integration process suit the current software product you are working on?	Process Suitability
18. Do the people that use Continuous Integration in your organisation follow any industry standard processes or organisation-defined processes when using the Continuous Integration process? (please describe how you work) 19. Do you think that your organisation has provided sufficient training or guidance on how to efficiently work within a continuous integration environment? (how has that affected the way you work with CI) 20. Do you believe that Continuous Integration is seen as a difficult process to adopt within your organisation? (how so/in what way)	Organizational
21. Would you consider the experience and knowledge of your team to be adequate in order to adopt a healthy Continuous Integration process? (please elaborate) 22. How has the introduction of Continuous Integration impacted you and your teams daily work?	Team

23. How does management support the adoption of Continuous Integration in your team and/or organisation?	Management
RQ2: What are the challenges of software requirements breakdown in practice?	
• *Mention the beginning of requirements breakdown section* • *Ask follow-up questions to refine the granularity and detail of responses* 24. Are there any challenges that you or your team have faced or believe can potentially face when breaking down software requirements?	Open-ended question
25. Do you believe that the majority of your team's software requirements are of correct size? (how so) 26. Do you believe that the majority of your team's software requirements are easy to understand? (please elaborate)	Characteristics
27. Does your organisation follow any industry standard processes or organisation-defined processes when breaking down requirements? (please describe) 28. Does the requirements breakdown process suit the current product you are working on? (in what way) 29. Would you consider the experience and knowledge of you team to be adequate for the requirements breakdown technique used within your team? (how so) 30. Do you believe that breaking down software requirements is seen as a difficult task within your team and/or organisation? (how so/in what way)	Process & Product

31. Do you feel that management behavior has an impact on the success of breaking down requirements? (how so)	Management
32. From your experience, could the challenges of requirements breakdown be an obstacle to the implementation of a healthy continuous integration process? (how so) 33. Is there anything else you would like to add or share about your experiences with continuous integration or requirements breakdown in your organisation?	Wrap up questions

B

NVivo

Nodes	**Sources**	**References**
Continuous integration challenges	13	173
Code dependencies	5	7
Dead code	2	2
Integration coordination	4	5
Domain applicability	11	17
Process suitability	6	8
Product complexity	7	9
Mindset	10	34
Change old habits	7	7
Expose work intention	4	5
Scepticism	7	16
Software requirements	8	13
Deliver feature growth	2	3
Integration of big impact changes	1	1
Requirements breakdown	7	9
Testing	9	19
Implementation and test dependencies	2	2
Preserving quality	4	5
Too many manual tests	4	7
Unstable test cases	3	3

Tools & Infrastructure	13	49
Code review	6	8
Integration queue	2	2
Maturity	10	17
Regression feedback time	5	10
Test automation	6	7
Understanding	10	34
Different interpretations	2	4
Increased pressure	4	7
Unclear goals	5	12
Bottom-up approach	6	11
Software requirements breakdown challenges	13	74
Alignment of requirements and tests	4	5
Implementation and test dependencies	4	5
Customer value	7	10
Access to customer	4	6
Delivering customer value	4	4
Guiding principles	11	32
Lack of guidance	5	8
No unified process	8	15
Ongoing responsibility shift	4	6
Unfit process	3	3
Requirements abstraction	12	27
Ambiguous requirements	4	6
Architectural design	4	4
Large requirements	6	10
Low level requirements	1	1
Product complexity	5	6
Software requirements breakdown's influence on continuous integration	12	36
Implications of software requirements breakdown	8	19
Implementation dependencies	3	4
Integration scope	4	7
Deliver feature growth	2	3

Delivery scope	1	2
Keep customer and technical perspective in sync	1	2
Test dependencies	2	3
Sustain testability	1	1
Unsuitable test cases	2	2
On demand software delivery (ODSD)	4	5
The necessity of software requirements breakdown	8	17
Big impact requirements	2	4
Maintenance versus feature development	2	2
Too large requirements	2	2